

PHP and Algorithmic Thinking for the Complete Beginner

Kindle Edition

Copyright © by Aristides S. Bouras and Loukia V. Ainarozidou

<http://www.bouraspage.com>

PHP is a copyright of the PHP Group.

The following are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries: Microsoft, Windows, IntelliSense, SQL Server, VBA, Visual Basic, and Visual C#.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, mechanical or electronic, including photocopying, recording, or by any information storage and retrieval system, without written permission from the authors.

Warning and Disclaimer

This book is designed to provide information about learning “Algorithmic Thinking,” mainly through the use of PHP scripting language. Every effort has been taken to make this book compatible with all previous releases of PHP, and it is almost certain to be compatible with any future releases of PHP.

The information is provided on an “as is” basis. The authors shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the files that may accompany it.

*This book is dedicated to
our adorable kids, Bill and Aphrodite,
as they "force" us to become better and better every day.*

Contents at a Glance

Preface.....	21
Section 1 Introductory Knowledge.....	27
<i>Chapter 1 How a Computer Works.....</i>	<i>28</i>
<i>Chapter 2 PHP.....</i>	<i>35</i>
<i>Chapter 3 Software Packages to Install</i>	<i>38</i>
<i>Review Questions in “Introductory Knowledge”</i>	<i>48</i>
Section 2 Getting Started with PHP	49
<i>Chapter 4 Introduction to Basic Algorithmic Concepts.....</i>	<i>50</i>
<i>Chapter 5 Variables and Constants</i>	<i>65</i>
<i>Chapter 6 Handling Input and Output.....</i>	<i>76</i>
<i>Chapter 7 Operators.....</i>	<i>83</i>
<i>Chapter 8 Trace Tables.....</i>	<i>98</i>
<i>Chapter 9 Using NetBeans IDE.....</i>	<i>107</i>
<i>Review Questions in “Getting Started with PHP”.....</i>	<i>123</i>
Section 3 Sequence Control Structures	125
<i>Chapter 10 Introduction to Sequence Control Structures</i>	<i>126</i>
<i>Chapter 11 Manipulating Numbers.....</i>	<i>135</i>
<i>Chapter 12 Complex Mathematical Expressions</i>	<i>145</i>
<i>Chapter 13 Exercises with a Quotient and a Remainder</i>	<i>152</i>
<i>Chapter 14 Manipulating Strings.....</i>	<i>163</i>
<i>Review Questions in “Sequence Control Structures”</i>	<i>177</i>
Section 4 Decision Control Structures.....	178
<i>Chapter 15 Introduction to Decision Control Structures</i>	<i>179</i>
<i>Chapter 16 The Single-Alternative Decision Structure</i>	<i>195</i>
<i>Chapter 17 The Dual-Alternative Decision Structure</i>	<i>205</i>
<i>Chapter 18 The Multiple-Alternative Decision Structure</i>	<i>218</i>
<i>Chapter 19 The Case Decision Structure.....</i>	<i>227</i>
<i>Chapter 20 Nested Decision Control Structures.....</i>	<i>237</i>
<i>Chapter 21 Tips and Tricks with Decision Control Structures.....</i>	<i>250</i>
<i>Chapter 22 Flowcharts with Decision Control Structures</i>	<i>295</i>

<i>Chapter 23 More Exercises with Decision Control Structures</i>	318
<i>Review Questions in “Decision Control Structures”</i>	364
Section 5 Loop Control Structures	365
<i>Chapter 24 Introduction to Loop Control Structures</i>	366
<i>Chapter 25 The Pre-Test Loop Structure</i>	371
<i>Chapter 26 The Post-Test Loop Structure</i>	388
<i>Chapter 27 Counted Loop Structures</i>	403
<i>Chapter 28 Nested Loop Control Structures</i>	424
<i>Chapter 29 Tips and Tricks with Loop Control Structures</i>	438
<i>Chapter 30 Flowcharts with Loop Control Structures</i>	488
<i>Chapter 31 More Exercises with Loop Control Structures</i>	514
<i>Review Questions in “Loop Control Structures”</i>	575
Section 6 Arrays	577
<i>Chapter 32 Introduction to Arrays</i>	578
<i>Chapter 33 One-Dimensional Arrays</i>	587
<i>Chapter 34 Two-Dimensional Arrays</i>	604
<i>Chapter 35 Tips and Tricks with Arrays</i>	632
<i>Chapter 36 Flowcharts with Arrays</i>	654
<i>Chapter 37 More Exercises with Arrays</i>	668
<i>Review Questions in “Arrays”</i>	767
Section 7 Subprograms	769
<i>Chapter 38 Introduction to Subprograms</i>	770
<i>Chapter 39 User-Defined Functions</i>	774
<i>Chapter 40 User-Defined Procedures</i>	788
<i>Chapter 41 Tips and Tricks with Subprograms</i>	800
<i>Chapter 42 Flowcharts with Subprograms</i>	838
<i>Chapter 43 More Exercises with Subprograms</i>	855
<i>Review Questions in “Subprograms”</i>	874
Index	875

Table of Contents

Preface.....21

 About the Authors 22

 Acknowledgments..... 23

 How This Book is Organized..... 23

 Who Should Buy This Book?..... 23

 Where to Find Answers to Review Questions and Exercises 24

 How to Report Errata..... 24

 Conventions Used in This Book..... 25

Section 1 Introductory Knowledge.....27

Chapter 1 How a Computer Works..... 28

 1.1 Introduction 29

 1.2 What is Hardware?..... 29

 1.3 What is Software? 30

 1.4 How a Computer Executes (Runs) a Program..... 30

 1.5 Compilers and Interpreters..... 30

 1.6 What is Source Code? 31

 1.7 Review Questions: True/False..... 31

 1.8 Review Questions: Multiple Choice 32

Chapter 2 PHP 35

 2.1 What is PHP?..... 36

 2.2 What is the Difference Between a Script and a Program? 36

 2.3 Why You Should Learn PHP 36

 2.4 How PHP Works 37

Chapter 3 Software Packages to Install 38

 3.1 XAMPP 39

 3.2 How to Set Up XAMPP..... 39

 3.3 NetBeans..... 41

 3.4 How to Set Up NetBeans IDE 42

Review Questions in “Introductory Knowledge” 48

Section 2 Getting Started with PHP 49

Chapter 4 Introduction to Basic Algorithmic Concepts..... 50

 4.1 What is an Algorithm? 51

 4.2 The Algorithm for Making a Cup of Tea 51

 4.3 Properties of an Algorithm 51

 4.4 Okay About Algorithms. But What is a Computer Program Anyway? 52

 4.5 The Party of Three! 52

4.6	The Three Main Stages Involved in Creating an Algorithm	52
4.7	Flowcharts	53
	Exercise 4.7-1 Finding the Average Value of Three Numbers	55
4.8	What are "Reserved Words"?	56
4.9	What is the Difference Between a Statement and a Command?	57
4.10	What is Structured Programming?	57
4.11	The Three Fundamental Control Structures.....	57
	Exercise 4.11-1 Understanding Control Structures Using Flowcharts.....	58
4.12	Your First PHP Script	59
4.13	What is the Difference Between Syntax Errors and Logic Errors?	59
4.14	Commenting Your Code	60
4.15	User-Friendly Programs and Scripts.....	61
4.16	Review Questions: True/False.....	61
4.17	Review Questions: Multiple Choice	63
Chapter 5 Variables and Constants		65
5.1	What is a Variable?.....	66
5.2	What is a Constant?.....	68
5.3	How Many Types of Variables and Constants Exist in PHP?	71
5.4	Rules for Naming Variables in PHP.....	71
5.5	Rules for Naming Constants in PHP.....	72
5.6	What Does the Phrase "Declare a Variable" Mean?	72
5.7	How to Declare Variables in PHP	73
5.8	How to Declare Constants in PHP.....	73
5.9	Review Questions: True/False.....	73
5.10	Review Questions: Multiple Choice	74
5.11	Review Exercises.....	75
Chapter 6 Handling Input and Output.....		76
6.1	Which Statement Outputs Messages and Results on a User's Screen?	77
6.2	How to Output Special Characters	78
6.3	Which Statement Lets the User Enter Data?.....	80
6.4	Review Questions: True/False.....	82
6.5	Review Questions: Multiple Choice	82
Chapter 7 Operators.....		83
7.1	The Value Assignment Operator	84
7.2	Arithmetic Operators	86
7.3	What is the Precedence of Arithmetic Operators?	87
7.4	Compound Assignment Operators	88
	Exercise 7.4-1 Which PHP Statements are Syntactically Correct?.....	89
	Exercise 7.4-2 Finding Variable Types.....	90
7.5	Incrementing/Decrementing Operators	90
7.6	String Operators	91

Exercise 7.6-1	Concatenating Names.....	92
7.7	Review Questions: True/False.....	92
7.8	Review Questions: Multiple Choice	93
7.9	Review Exercises.....	95
Chapter 8 Trace Tables.....		98
8.1	What is a Trace Table?.....	99
Exercise 8.1-1	Creating a Trace Table	100
Exercise 8.1-2	Swapping Values of Variables	100
Exercise 8.1-3	Swapping Values of Variables – A Second Approach	103
Exercise 8.1-4	Creating a Trace Table	103
Exercise 8.1-5	Creating a Trace Table	104
8.2	Review Questions: True/False.....	105
8.3	Review Exercises.....	105
Chapter 9 Using NetBeans IDE.....		107
9.1	Creating a New PHP Project.....	108
9.2	Writing and Executing a PHP Script	110
9.3	What "Debugging" Means.....	114
9.4	Debugging PHP Scripts with NetBeans IDE	114
9.5	Review Exercises.....	121
Review Questions in "Getting Started with PHP"		123
Section 3 Sequence Control Structures		125
Chapter 10 Introduction to Sequence Control Structures		126
10.1	What is the Sequence Control Structure?	127
Exercise 10.1-1	Calculating the Area of a Parallelogram	127
Exercise 10.1-2	Calculating the Area of a Circle	128
Exercise 10.1-3	Calculating Fuel Economy	128
Exercise 10.1-4	Where is the Car? Calculating Distance Traveled	129
Exercise 10.1-5	Kelvin to Fahrenheit	130
Exercise 10.1-6	Calculating Sales Tax.....	131
Exercise 10.1-7	Calculating a Sales Discount.....	131
Exercise 10.1-8	Calculating the Sales Tax Rate and Discount.....	132
10.2	Review Exercises.....	133
Chapter 11 Manipulating Numbers		135
11.1	Introduction	136
11.2	Useful Mathematical Functions	136
Exercise 11.2-1	Calculating the Distance Between Two Points.....	140
Exercise 11.2-2	How Far Did the Car Travel?.....	141
11.3	Review Questions: True/False.....	142
11.4	Review Questions: Multiple Choice	142
11.5	Review Exercises.....	143
Chapter 12 Complex Mathematical Expressions		145

12.1	Writing Complex Mathematical Expressions	146
Exercise 12.1-1	Representing Mathematical Expressions in PHP	146
Exercise 12.1-2	Writing a Mathematical Expression in PHP	147
Exercise 12.1-3	Writing a Complex Mathematical Expression in PHP	147
12.2	Review Exercises.....	149

Chapter 13 Exercises with a Quotient and a Remainder 152

13.1	Introduction	153
Exercise 13.1-1	Calculating the Quotient and Remainder of Integer Division	153
Exercise 13.1-2	Finding the Sum of Digits.....	154
Exercise 13.1-3	Displaying an Elapsed Time	159
Exercise 13.1-4	Reversing a Number.....	160
13.2	Review Exercises.....	161

Chapter 14 Manipulating Strings..... 163

14.1	Introduction	164
14.2	The Position of a Character in a String	164
14.3	Retrieving an Individual Character From a String	165
Exercise 14.3-1	Displaying a String Backwards	166
14.4	Useful String Functions	166
Exercise 14.4-1	Switching the Order of Names	170
Exercise 14.4-2	Creating a Login ID	171
Exercise 14.4-3	Creating a Random Word	172
14.5	Review Questions: True/False.....	173
14.6	Review Questions: Multiple Choice	173
14.7	Review Exercises.....	175

Review Questions in “Sequence Control Structures” 177

Section 4 Decision Control Structures.....178

Chapter 15 Introduction to Decision Control Structures 179

15.1	What is a Decision Control Structure?	180
15.2	What is a Boolean Expression?	180
15.3	How to Write Boolean Expressions	180
Exercise 15.3-1	Filling in the Table	181
15.4	Logical Operators and Complex Boolean Expressions	182
15.5	What is the Order of Precedence of Logical Operators?	183
15.6	What is the Order of Precedence of Arithmetic, Comparison, and Logical Operators?.....	183
Exercise 15.6-1	Filling in the Truth Table	184
Exercise 15.6-2	Calculating the Results of Complex Boolean Expressions	185
Exercise 15.6-3	Converting English Sentences to Boolean Expressions	186
15.7	How to Negate Boolean Expressions	188
Exercise 15.7-1	Negating Boolean Expressions.....	189
15.8	Review Questions: True/False.....	190

15.9	Review Questions: Multiple Choice	191
15.10	Review Exercises.....	192
Chapter 16 The Single-Alternative Decision Structure		195
16.1	The Single-Alternative Decision Structure	196
	Exercise 16.1-1 Trace Tables and Single-Alternative Decision Structures	197
	Exercise 16.1-2 The Absolute Value of a Number	198
16.2	Review Questions: True/False.....	199
16.3	Review Questions: Multiple Choice	200
16.4	Review Exercises.....	201
Chapter 17 The Dual-Alternative Decision Structure		205
17.1	The Dual-Alternative Decision Structure	206
	Exercise 17.1-1 Finding the Output Message.....	206
	Exercise 17.1-2 Trace Tables and Dual-Alternative Decision Structures	207
	Exercise 17.1-3 Who is the Greatest?	209
	Exercise 17.1-4 Finding Odd and Even Numbers	211
	Exercise 17.1-5 Weekly Wages.....	212
17.2	Review Questions: True/False.....	213
17.3	Review Questions: Multiple Choice	213
17.4	Review Exercises.....	214
Chapter 18 The Multiple-Alternative Decision Structure		218
18.1	The Multiple-Alternative Decision Structure	219
	Exercise 18.1-1 Trace Tables and Multiple-Alternative Decision Structures	220
	Exercise 18.1-2 Counting the Digits	222
18.2	Review Questions: True/False.....	223
18.3	Review Exercises.....	224
Chapter 19 The Case Decision Structure.....		227
19.1	The Case Decision Structure	228
	Exercise 19.1-1 The Days of the Week.....	230
19.2	Review Questions: True/False.....	232
19.3	Review Exercises.....	233
Chapter 20 Nested Decision Control Structures.....		237
20.1	What are Nested Decision Control Structures?	238
	Exercise 20.1-1 Trace Tables and Nested Decision Control Structures.....	239
	Exercise 20.1-2 Positive, Negative or Zero?	241
20.2	A Mistake That You Will Probably Make!.....	242
20.3	Review Questions: True/False.....	246
20.4	Review Exercises.....	246
Chapter 21 Tips and Tricks with Decision Control Structures		250
21.1	Introduction	251
21.2	Choosing a Decision Control Structure	251
21.3	Streamlining the Decision Control Structure	251

	Exercise 21.3-1 “Shrinking” the Algorithm	253
	Exercise 21.3-2 “Shrinking” the PHP Script.....	254
	Exercise 21.3-3 “Shrinking” the Algorithm	255
21.4	Logical Operators – to Use, or not to Use: That is the Question!.....	258
	Exercise 21.4-1 Rewriting the Code	259
	Exercise 21.4-2 Rewriting the Code	260
21.5	Merging Two or More Single-Alternative Decision Structures	261
	Exercise 21.5-1 Merging the Decision Control Structures	262
	Exercise 21.5-2 Merging the Decision Control Structures	263
21.6	Replacing Two Single-Alternative Decision Structures with a Dual- Alternative One	265
	Exercise 21.6-1 “Merging” the Decision Control Structures.....	266
21.7	Put the Boolean Expressions Most Likely to be True First.....	267
	Exercise 21.7-1 Rearranging the Boolean Expressions	268
21.8	Converting a Case Decision Structure to a Multiple-Alternative Decision Structure, and Vice Versa	269
	Exercise 21.8-1 Converting the PHP Script.....	270
	Exercise 21.8-2 Converting the PHP Script.....	271
	Exercise 21.8-3 Converting the PHP Script.....	273
21.9	Converting a Multiple-Alternative Decision Structure to Nested Decision Control Structures, and Vice Versa	273
	Exercise 21.9-1 Converting the PHP Script.....	274
	Exercise 21.9-2 Converting the PHP Script.....	276
21.10	Converting a Case Decision Structure to Nested Decision Control Structures, and Vice Versa	277
	Exercise 21.10-1 Converting the PHP Script.....	278
	Exercise 21.10-2 Converting the PHP Script.....	279
21.11	What is Code Indentation and Why is it so Important?	280
21.12	Using the “From Inner to Outer” Method in Decision Control Structures .	281
21.13	Review Questions: True/False.....	283
21.14	Review Questions: Multiple Choice	284
21.15	Review Exercises.....	287
Chapter 22 Flowcharts with Decision Control Structures		295
22.1	Introduction	296
22.2	Converting PHP Scripts to Flowcharts	296
	Exercise 22.2-1 Designing the Flowchart	297
	Exercise 22.2-2 Designing the Flowchart	298
	Exercise 22.2-3 Designing the Flowchart	299
	Exercise 22.2-4 Designing the Flowchart	301
22.3	Converting Flowcharts to PHP Scripts	302
	Exercise 22.3-1 Writing the PHP Script.....	303
	Exercise 22.3-2 Writing the PHP Script.....	305
	Exercise 22.3-3 Writing the PHP Script.....	306

	Exercise 22.3-4	Writing the PHP Script.....	307
	Exercise 22.3-5	Writing the PHP Script.....	310
22.4	Review Exercises.....		311
Chapter 23 More Exercises with Decision Control Structures.....			318
23.1	Simple Exercises with Decision Control Structures		319
	Exercise 23.1-1	Both Odds or Both Evens?	319
	Exercise 23.1-2	Validating Data Input and Finding if a Number is Exactly Divisible by both 5 and 8?	319
	Exercise 23.1-3	Is it an Integer?	321
	Exercise 23.1-4	Converting Gallons to Liters, and Vice Versa	323
	Exercise 23.1-5	Converting Gallons to Liters, and Vice Versa (with Data Validation)	324
	Exercise 23.1-6	Where is the Tollkeeper?	325
	Exercise 23.1-7	The Most Scientific Calculator Ever!	326
23.2	Decision Control Structures in Solving Mathematical Problems		328
	Exercise 23.2-1	Finding the Value of y	328
	Exercise 23.2-2	Finding the Values of y	329
	Exercise 23.2-3	Validating Data Input and Finding the Values of y	330
	Exercise 23.2-4	Solving the Linear Equation $ax + b = 0$	331
	Exercise 23.2-5	Solving the Quadratic Equation $ax^2 + bx + c = 0$	333
23.3	Finding Minimum and Maximum Values with Decision Control Structures.....		335
	Exercise 23.3-1	Finding the Name of the Heaviest Person	338
23.4	Exercises with Series of Consecutive Ranges of Values		339
	Exercise 23.4-1	Calculating the Discount	339
	Exercise 23.4-2	Validating Data Input and Calculating the Discount.....	341
	Exercise 23.4-3	Sending a Parcel	343
	Exercise 23.4-4	Finding the Values of y	345
	Exercise 23.4-5	Progressive Rates and Electricity Consumption	348
	Exercise 23.4-6	Progressive Rates, Electricity Consumption, Taxes, Data Validation and Code Optimization, All in One!	349
	Exercise 23.4-7	Progressive Rates and Text Messaging Services	352
23.5	Exercises of a General Nature with Decision Control Structures		353
	Exercise 23.5-1	Finding a Leap Year	353
	Exercise 23.5-2	Displaying the Days of the Month.....	354
	Exercise 23.5-3	Is the Number a Palindrome?	356
	Exercise 23.5-4	Checking for Proper Capitalization and Punctuation	357
23.6	Review Exercises.....		359
Review Questions in "Decision Control Structures"			364
Section 5 Loop Control Structures			365
Chapter 24 Introduction to Loop Control Structures			366
24.1	What is a Loop Control Structure?		367
24.2	From Sequence Control to Loop Control Structures		367
24.3	Review Questions: True/False.....		369

Chapter 25 The Pre-Test Loop Structure.....	371
25.1 The Pre-Test Loop Structure	372
Exercise 25.1-1 Designing the Flowchart and Counting the Total Number of Iterations	373
Exercise 25.1-2 Counting the Total Number of Iterations.....	375
Exercise 25.1-3 Designing the Flowchart and Counting the Total Number of Iterations	375
Exercise 25.1-4 Counting the Total Number of Iterations.....	376
Exercise 25.1-5 Finding the Sum of 10 Numbers.....	377
Exercise 25.1-6 Finding the Product of 20 Numbers	378
Exercise 25.1-7 Finding the Product of N Numbers	379
Exercise 25.1-8 Finding the Sum of Odd Numbers	380
Exercise 25.1-9 Finding the Sum of an Unknown Quantity of Numbers.....	381
25.2 Review Questions: True/False.....	383
25.3 Review Questions: Multiple Choice	383
25.4 Review Exercises.....	385
Chapter 26 The Post-Test Loop Structure.....	388
26.1 The Post-Test Loop Structure	389
Exercise 26.1-1 Designing the Flowchart and Counting the Total Number of Iterations	390
Exercise 26.1-2 Counting the Total Number of Iterations.....	391
Exercise 26.1-3 Designing the Flowchart and Counting the Total Number of Iterations	392
Exercise 26.1-4 Counting the Total Number of Iterations.....	394
Exercise 26.1-5 Finding the Product of N Numbers	394
Exercise 26.1-6 Finding the Product of an Unknown Quantity of Numbers.....	395
26.2 Review Questions: True/False.....	396
26.3 Review Questions: Multiple Choice	397
26.4 Review Exercises.....	399
Chapter 27 Counted Loop Structures	403
27.1 Counted Loop Structures.....	404
Exercise 27.1-1 Designing the Flowchart and Creating the Trace Table.....	407
Exercise 27.1-2 Creating the Trace Table.....	409
Exercise 27.1-3 Counting the Total Number of Iterations.....	410
Exercise 27.1-4 Finding the Sum of 10 Numbers.....	411
Exercise 27.1-5 Finding the Square Roots from 0 to N	411
27.2 Rules that Apply to Counted Loop Structures.....	412
Exercise 27.2-1 Counting the Total Number of Iterations.....	412
Exercise 27.2-2 Counting the Total Number of Iterations.....	413
Exercise 27.2-3 Counting the Total Number of Iterations.....	414
Exercise 27.2-4 Counting the Total Number of Iterations.....	415
Exercise 27.2-5 Finding the Average Value of N Numbers.....	416
27.3 Review Questions: True/False.....	417
27.4 Review Questions: Multiple Choice	418

27.5	Review Exercises.....	421
Chapter 28 Nested Loop Control Structures.....		424
28.1	What is a Nested Loop?	425
	Exercise 28.1-1 Say “Hello Zeus”. Designing the Flowchart and Counting the Total Number of Iterations.....	426
	Exercise 28.1-2 Creating the Trace Table.....	427
28.2	Rules that Apply to Nested Loops.....	429
	Exercise 28.2-1 Breaking the First Rule.....	429
	Exercise 28.2-2 Counting the Total Number of Iterations.....	430
28.3	Review Questions: True/False.....	431
28.4	Review Questions: Multiple Choice	432
28.5	Review Exercises.....	434
Chapter 29 Tips and Tricks with Loop Control Structures		438
29.1	Introduction	439
29.2	Choosing a Loop Control Structure.....	439
29.3	The “Ultimate” Rule	439
29.4	Breaking Out of a Loop	443
29.5	Cleaning Out Your Loops	446
	Exercise 29.5-1 Cleaning Out the Loop.....	447
	Exercise 29.5-2 Cleaning Out the Loop.....	448
29.6	Endless Loops and How to Avoid Them	449
29.7	Converting from a Counted Loop Structure to a Pre-Test Loop Structure	450
	Exercise 29.7-1 Converting the PHP Script.....	450
	Exercise 29.7-2 Converting the PHP Script.....	452
29.8	Converting from a Pre-Test Loop Structure to a Counted Loop Structure	453
	Exercise 29.8-1 Converting the PHP Script.....	454
	Exercise 29.8-2 Converting the PHP Script.....	455
	Exercise 29.8-3 Converting the PHP Script.....	456
	Exercise 29.8-4 Converting the PHP Script.....	458
	Exercise 29.8-5 Converting the PHP Script.....	459
29.9	Converting from a Post-Test Loop Structure to a Pre-Test Loop Structure.....	461
	Exercise 29.9-1 Converting the PHP Script.....	461
	Exercise 29.9-2 Converting the PHP Script.....	463
	Exercise 29.9-3 Converting the PHP Script.....	464
29.10	Converting from a Pre-Test Loop Structure to a Post-Test Loop Structure.....	465
	Exercise 29.10-1 Converting the PHP Script.....	466
	Exercise 29.10-2 Converting the PHP Script.....	467
	Exercise 29.10-3 Converting the PHP Script.....	467
29.11	Converting from a Counted Loop Structure to a Post-Test Loop Structure.....	469
	Exercise 29.11-1 Converting the PHP Script.....	470
	Exercise 29.11-2 Converting the PHP Script.....	471
29.12	Converting from a Post-Test Loop Structure to a Counted Loop Structure.....	472
	Exercise 29.12-1 Converting the PHP Script.....	474

Exercise 29.12-2	Converting the PHP Script.....	475
29.13	Using the “From Inner to Outer” Method in Loop Control Structures	476
29.14	Review Questions: True/False.....	478
29.15	Review Questions: Multiple Choice	480
29.16	Review Exercises.....	482
Chapter 30 Flowcharts with Loop Control Structures		488
30.1	Introduction	489
30.2	Converting PHP Scripts to Flowcharts	489
Exercise 30.2-1	Designing the Flowchart.....	490
Exercise 30.2-2	Designing the Flowchart.....	491
Exercise 30.2-3	Designing the Flowchart.....	492
Exercise 30.2-4	Designing the Flowchart.....	493
Exercise 30.2-5	Designing the Flowchart.....	495
Exercise 30.2-6	Designing the Flowchart.....	496
Exercise 30.2-7	Designing the Flowchart.....	497
30.3	Converting Flowcharts to PHP Scripts	498
Exercise 30.3-1	Writing the PHP Script.....	499
Exercise 30.3-2	Writing the PHP Script.....	500
Exercise 30.3-3	Writing the PHP Script.....	501
Exercise 30.3-4	Writing the PHP Script.....	503
30.4	Review Exercises.....	507
Chapter 31 More Exercises with Loop Control Structures		514
31.1	Simple Exercises with Loop Control Structures.....	515
Exercise 31.1-1	Finding the Sum of $1 + 2 + 3 + \dots + 100$	515
Exercise 31.1-2	Finding the Product of $2 \times 4 \times 6 \times 8 \times 10$	516
Exercise 31.1-3	Finding the Sum of $2^2 + 4^2 + 6^2 + \dots (2N)^2$	517
Exercise 31.1-4	Finding the Sum of $3^3 + 6^6 + 9^9 + \dots (3N)^{3N}$	518
Exercise 31.1-5	Finding the Average Value of Positive Numbers.....	518
Exercise 31.1-6	Counting the Numbers According to Which is Greater	519
Exercise 31.1-7	Counting the Numbers According to Their Digits.....	520
Exercise 31.1-8	How Many Numbers Fit in a Sum	521
Exercise 31.1-9	Finding the Sum of Integers	522
Exercise 31.1-10	Iterating as Many Times as the User Wishes	523
Exercise 31.1-11	Finding the Sum of the Digits.....	524
Exercise 31.1-12	Counting the Digits	526
31.2	Exercises with Nested Loop Control Structures	527
Exercise 31.2-1	Displaying all Three-Digit Integers that Contain a Given Digit ..	527
Exercise 31.2-2	Displaying all Instances of a Specified Condition.....	529
31.3	Data Validation with Loop Control Structures.....	530
Exercise 31.3-1	Finding the Square Root - Validation Without Error Messages..	532
Exercise 31.3-2	Finding the Square Root - Validation with One Error Message ..	533
Exercise 31.3-3	Finding the Square Root - Validation with Individual Error Messages.....	534
Exercise 31.3-4	Finding the Sum of 10 Numbers.....	534

31.4	Using Loop Control Structures to Solve Mathematical Problems	536
Exercise 31.4-1	Calculating the Area of as Many Triangles as the User Wishes	536
Exercise 31.4-2	Finding x and y	537
Exercise 31.4-3	From Russia with Love	538
Exercise 31.4-4	Finding the Number of Divisors	541
Exercise 31.4-5	Is the Number a Prime?	543
Exercise 31.4-6	Finding all Prime Numbers from 1 to N	544
Exercise 31.4-7	Heron's Square Root	546
Exercise 31.4-8	Calculating π	548
Exercise 31.4-9	Approximating a Real with a Fraction	549
31.5	Finding Minimum and Maximum Values with Loop Control Structures ...	550
Exercise 31.5-1	Validating and Finding the Minimum and the Maximum Value	553
Exercise 31.5-2	Validating and Finding the Maximum Temperature	554
Exercise 31.5-3	"Making the Grade"	556
31.6	Exercises of a General Nature with Loop Control Structures	558
Exercise 31.6-1	Fahrenheit to Kelvin, from 0 to 100	558
Exercise 31.6-2	Wheat on a Chessboard	559
Exercise 31.6-3	Just a Poll	560
Exercise 31.6-4	Is the Message a Palindrome?	561
31.7	Review Questions: True/False	566
31.8	Review Exercises	568
	Review Questions in "Loop Control Structures"	575

Section 6 Arrays

Chapter 32 Introduction to Arrays

32.1	Introduction	579
32.2	What is an Array?	580
Exercise 32.2-1	Designing an Array	583
Exercise 32.2-2	Designing Arrays	584
Exercise 32.2-3	Designing Arrays	585
32.3	Review Questions: True/False	585
32.4	Review Exercises	586

Chapter 33 One-Dimensional Arrays

33.1	Creating One-Dimensional Arrays in PHP	588
33.2	How to Get Values from One-Dimensional Arrays	589
Exercise 33.2-1	Creating the Trace Table	590
Exercise 33.2-2	Using a Non-Existing Index	590
33.3	How to Add Values Entered by the User to a One-Dimensional Array	591
33.4	How to Iterate Through a One-Dimensional Array	591
Exercise 33.4-1	Displaying Words in Reverse Order	592
Exercise 33.4-2	Displaying Positive Numbers in Reverse Order	593
Exercise 33.4-3	Displaying Even Numbers in Odd-Numbered Index Positions	594
Exercise 33.4-4	Finding the Sum	595
33.5	Review Questions: True/False	596

33.6 Review Questions: Multiple Choice 598

33.7 Review Exercises..... 600

Chapter 34 Two-Dimensional Arrays..... 604

34.1 Creating Two-Dimensional Arrays in PHP 605

34.2 How to Get Values from Two-Dimensional Arrays..... 608

Exercise 34.2-1 Creating the Trace Table.....609

34.3 How to Add Values Entered by the User to a Two-Dimensional Array 610

34.4 How to Iterate Through a Two-Dimensional Array..... 611

Exercise 34.4-1 Displaying Reals Only.....614

Exercise 34.4-2 Displaying Odd Columns Only.....615

34.5 What’s the Story on Variables \$i and \$j? 616

34.6 Square Arrays 616

Exercise 34.6-1 Finding the Sum of the Elements of the Main Diagonal.....616

Exercise 34.6-2 Finding the Sum of the Elements of the Antidiagonal 618

Exercise 34.6-3 Filling in the Array 620

34.7 Review Questions: True/False..... 621

34.8 Review Questions: Multiple Choice 624

34.9 Review Exercises..... 627

Chapter 35 Tips and Tricks with Arrays 632

35.1 Introduction 633

35.2 Processing Each Row Individually 633

Exercise 35.2-1 Finding the Average Value 635

35.3 Processing Each Column Individually 637

Exercise 35.3-1 Finding the Average Value 639

35.4 How to Use One-Dimensional Along with Two-Dimensional Arrays 642

Exercise 35.4-1 Finding the Average Value 642

35.5 Creating a One-Dimensional Array from a Two-Dimensional Array 645

35.6 Creating a Two-Dimensional Array from a One-Dimensional Array 647

35.7 Review Questions: True/False..... 648

35.8 Review Questions: Multiple Choice 649

35.9 Review Exercises..... 650

Chapter 36 Flowcharts with Arrays..... 654

36.1 Introduction 655

36.2 Converting PHP Scripts to Flowcharts 655

Exercise 36.2-1 Designing the Flowchart.....655

Exercise 36.2-2 Designing the Flowchart.....656

Exercise 36.2-3 Designing the Flowchart.....657

36.3 Converting Flowcharts to PHP Scripts 659

Exercise 36.3-1 Writing the PHP Script.....659

Exercise 36.3-2 Writing the PHP Script.....660

Exercise 36.3-3 Writing the PHP Script.....661

36.4 Review Exercises..... 663

Chapter 37 More Exercises with Arrays 668

37.1 Simple Exercises with Arrays..... 669

 Exercise 37.1-1 Creating an Array that Contains the Average Values of its
 Neighboring Elements.....669

 Exercise 37.1-2 Creating an Array with the Greatest Values 671

 Exercise 37.1-3 Merging One-Dimensional Arrays.....673

 Exercise 37.1-4 Merging Two-Dimensional Arrays.....675

 Exercise 37.1-5 Creating Two Arrays – Separating Positive from Negative
 Values.....676

 Exercise 37.1-6 Creating an Array with Those who Contain Digit 5 680

37.2 Data Validation with Arrays..... 681

 Exercise 37.2-1 Displaying Odds in Reverse Order – Validation Without Error
 Messages.....683

 Exercise 37.2-2 Displaying Odds in Reverse Order – Validation with One Error
 Message.....684

 Exercise 37.2-3 Displaying Odds in Reverse Order – Validation with Individual
 Error Messages.....685

37.3 Finding Minimum and Maximum Values in Arrays 686

 Exercise 37.3-1 Which Depth is the Greatest?.....686

 Exercise 37.3-2 Which Lake is the Deepest? 688

 Exercise 37.3-3 Which Lake, in Which Country, Having Which Average Area, is
 the Deepest?.....688

 Exercise 37.3-4 Which Students are the Tallest?.....691

 Exercise 37.3-5 Finding the Minimum Value of a Two-Dimensional Array.....692

 Exercise 37.3-6 Finding the City with the Coldest Day 694

 Exercise 37.3-7 Finding the Minimum and the Maximum Value of Each Row.....696

 Exercise 37.3-8 Finding the Minimum and the Maximum Value of Each Column699

37.4 Sorting Arrays 702

 Exercise 37.4-1 The Bubble Sort Algorithm – Sorting One-Dimensional Arrays
 with Numeric Values.....703

 Exercise 37.4-2 Sorting One-Dimensional Arrays with Alphanumeric Values.....708

 Exercise 37.4-3 Sorting One-Dimensional Arrays While Preserving the
 Relationship with a Second Array 709

 Exercise 37.4-4 Sorting Last and First Names.....710

 Exercise 37.4-5 Sorting a Two-Dimensional Array.....713

 Exercise 37.4-6 The Modified Bubble Sort Algorithm – Sorting One-Dimensional
 Arrays.....714

 Exercise 37.4-7 The Five Best Scorers 716

 Exercise 37.4-8 The Selection Sort Algorithm – Sorting One-Dimensional Arrays720

 Exercise 37.4-9 Sorting One-Dimensional Arrays While Preserving the
 Relationship with a Second Array 722

 Exercise 37.4-10 The Insertion Sort Algorithm – Sorting One-Dimensional Arrays723

 Exercise 37.4-11 The Three Worst Elapsed Times.....726

37.5 Searching Elements in Arrays 728

Exercise 37.5-1	The Linear Search Algorithm – Searching in a One-Dimensional Array that may Contain the Same Value Multiple Times	729
Exercise 37.5-2	Display the Last Names of All Those People Who Have the Same First Name.....	729
Exercise 37.5-3	Searching in a One-Dimensional Array that Contains Unique Values	731
Exercise 37.5-4	Searching for a Given Social Security Number	733
Exercise 37.5-5	Searching in a Two-Dimensional Array that may Contain the Same Value Multiple Times	734
Exercise 37.5-6	Searching in a Two-Dimensional Array that Contains Unique Values	736
Exercise 37.5-7	Checking if a Value Exists in all Columns.....	738
Exercise 37.5-8	The Binary Search Algorithm – Searching in a Sorted One-Dimensional Array.....	741
Exercise 37.5-9	Display all the Historical Events for a Country	743
Exercise 37.5-10	Searching in Each Column of a Two-Dimensional Array.....	745
37.6	Exercises of a General Nature with Arrays.....	749
Exercise 37.6-1	On Which Days was There a Possibility of Snow?	749
Exercise 37.6-2	Was There Any Possibility of Snow?	750
Exercise 37.6-3	In Which Cities was There a Possibility of Snow?	752
Exercise 37.6-4	Display from Highest to Lowest Grades by Student, and in Alphabetical Order.....	755
Exercise 37.6-5	Archery at the Summer Olympics.....	757
37.7	Review Questions: True/False.....	759
37.8	Review Exercises.....	761

Review Questions in “Arrays” **767**

Section 7 Subprograms **769**

Chapter 38 Introduction to Subprograms **770**

38.1	What is Procedural Programming?	771
38.2	What is Modular Programming?	772
38.3	What Exactly is a Subprogram?	772
38.4	Review Questions: True/False.....	773

Chapter 39 User-Defined Functions..... **774**

39.1	Writing your Own Functions in PHP	775
39.2	How Do You Call a Function?	775
39.3	Formal and Actual Function Arguments.....	778
39.4	How Does a Function Execute?	779
Exercise 39.4-1	Back to Basics – Calculating the Sum of Two Numbers	780
Exercise 39.4-2	Calculating the Sum of Two Numbers Using Fewer Lines of Code!	782
39.5	Review Questions: True/False.....	782
39.6	Review Exercises.....	784

Chapter 40 User-Defined Procedures..... 788

40.1 Writing your Own Procedures in PHP 789

40.2 How Do You Call a Procedure?..... 790

40.3 Formal and Actual Procedure Arguments 791

40.4 How Does a Procedure Execute?..... 792

Exercise 40.4-1 Back to Basics – Displaying the Absolute Value of a Number 793

Exercise 40.4-2 A Simple Currency Converter 795

40.5 Review Questions: True/False..... 796

40.6 Review Exercises..... 797

Chapter 41 Tips and Tricks with Subprograms..... 800

41.1 Can Two Subprograms use Variables of the Same Name?..... 801

41.2 Can a Subprogram Call Another Subprogram?..... 802

Exercise 41.2-1 A Currency Converter – Using Functions with Procedures 803

41.3 Passing Arguments by Value and by Reference 805

Exercise 41.3-1 Finding the Logic Error.....808

Exercise 41.3-2 Finding the Absolute Value of a Number 809

Exercise 41.3-3 Converting a Function to a Procedure.....811

41.4 Passing and/or Returning an Array..... 812

41.5 Default Argument Values (Optional Arguments) 815

41.6 The Scope of a Variable 816

41.7 Converting Parts of Code into Subprograms 818

41.8 Recursion..... 823

Exercise 41.8-1 Calculating the Fibonacci Sequence Recursively 825

41.9 Review Questions: True/False..... 828

41.10 Review Exercises..... 829

Chapter 42 Flowcharts with Subprograms..... 838

42.1 Designing and Calling Sub-Algorithms in Flowcharts 839

42.2 Converting PHP Scripts to Flowcharts 841

Exercise 42.2-1 Designing the Flowchart 841

Exercise 42.2-2 Designing the Flowchart 842

Exercise 42.2-3 Designing the Flowchart 843

42.3 Converting Flowcharts to PHP Scripts 845

Exercise 42.3-1 Writing the PHP Script.....845

Exercise 42.3-2 Writing the PHP Script.....846

42.4 Review Exercises..... 848

Chapter 43 More Exercises with Subprograms..... 855

43.1 Simple Exercises with Subprograms..... 856

Exercise 43.1-1 Finding the Average Values of Positive Integers.....856

Exercise 43.1-2 Finding the Sum of Odd Positive Integers 857

Exercise 43.1-3 Finding the Values of y.....858

Exercise 43.1-4 Roll, Roll, Roll the... Dice!.....859

Exercise 43.1-5 How Many Times Does Each Number of the Dice Appear?.....860

43.2 Exercises of a General Nature with Subprograms..... 863

Exercise 43.2-1 Validating Data Input.....863

Exercise 43.2-2 Sorting an Array.....865

Exercise 43.2-3 Progressive Rates and Electricity Consumption 867

43.3 Review Exercises..... 868

Review Questions in “Subprograms” 874

Index.....875

Preface

About the Authors

Aristides S. Bouras

Aristides¹ S. Bouras was born in 1973. During his early childhood, he discovered a love of computer programming. He got his first computer at the age of 12, a Commodore 64, which incorporated a ROM-based version of the BASIC programming language and 64 kilobytes of RAM!!!

He holds a degree in Computer Engineering from the Technological Educational Institute of Piraeus, and a degree in Electrical and Computer Engineering from the Democritus Polytechnic University of Thrace.

He worked as a software developer at a company that specialized in industrial data flow and labelling of products. His main job was to develop software applications for data terminals (originally in TALL and later in VB.NET language), as well as PC software applications for collecting and storing data on a Microsoft SQL Server.

He has developed many applications such as warehouse managing systems and websites for companies and other organizations. Nowadays, he works as a high school teacher. He mainly teaches courses in computer networks, programming tools for the Internet/intranets, and databases.

He is married to Loukia V. Ainarozidou and they have two children.

Loukia V. Ainarozidou

Loukia V. Ainarozidou was born in 1975. She got her first computer at the age of 13, an Amstrad CPC6128 with 128 kilobytes of RAM and an internal 3-inch floppy disk drive!!!

She holds a degree in Computer Engineering from the Technological Educational Institute of Piraeus, and a degree in Electrical and Computer Engineering from the Democritus Polytechnic University of Thrace.

She worked as a supervisor in the data logistics department of a company involved in the packaging of fruit and vegetables. Nowadays, she works as a high school teacher. She mainly teaches courses in computer networks, computer programming, and digital design.

She is married to Aristides S. Bouras and they have two children.

¹ Aristides (530 BC–468 BC) was an ancient Athenian statesman and general. The ancient historian Herodotus cited him as “the best and most honorable man in Athens.” He was so fair in all that he did that he was often referred to as “Aristides the Just.” He flourished in the early quarter of Athens’s Classical period and helped Athenians defeat the Persians at the battles of Salamis and Plataea.

Acknowledgments

If it weren't for Dr. Yannis T. Kappos, we may never have written this book. As a renowned author of technical books on AutoCAD, he inspired us to take a seat and start writing. We wish to express our enormous appreciation to him for generously spending his time answering all of our questions—even the foolish ones.

We would also like to extend our thanks, with particular gratefulness, to our friend and senior editor Victoria (Vicki) Austin for her assistance in copy editing. Without her, this book might not have reached its full potential. With her patient guidance and valuable and constructive suggestions, she helped us bring this book up to a higher level!

How This Book is Organized

The book you hold in your hands follows the spiral curriculum teaching approach, a method proposed in 1960 by Jerome Bruner, an American psychologist. According to this method, as a subject is being taught, basic ideas are revisited at intervals—at a more sophisticated level each time—until the reader achieves a complete understanding of the subject. First, the reader learns the basic elements without worrying about the details. Later, more details are taught and basic elements are mentioned again and again, eventually being stored in the brain's long term memory.

According to Jerome Bruner, learning requires the student's active participation, experimentation, exploration, and discovery. This book contains many examples, most of which can be practically performed. This gives the reader the opportunity to get his or her hands on PHP and become capable of creating his or her own programs.

Who Should Buy This Book?

This book is for anyone who wants to learn computer programming and knows absolutely nothing about it. Of course, if you are wondering whether this book is going to teach you how to create amazing websites or incredible applications, the answer is “no”—that is a job for other books. So many books out there can teach you those skills in PHP, Java, C++, or C#. Many of them even claim that they can teach you in 24 hours! Don't laugh! They probably can do that, but all of them take one thing for granted—that the reader knows some basics about computer programming. None of those books, unfortunately, bothers to teach you the first thing that a novice programmer needs to learn, which is “Algorithmic Thinking.”

Algorithmic Thinking involves more than just learning code. It is a problem solving process that involves learning *how to* code. With over 800 pages, and containing more than 300 solved and 400 unsolved exercises, over 450 true/false, 150 multiple choice, and 180 review questions (the solutions and the answers to which can be found on the Internet), this book is ideal for students, teachers, professors, novices or average programmers, or for anyone who wants to start learning or teaching computer programming using the proper conventions and techniques.

Where to Find Answers to Review Questions and Exercises

Answers to all of the review questions, as well as the solutions to all review exercises, are available free of charge on the Internet. You can download them from the following address:

<http://www.bouraspage.com>

How to Report Errata

Although we have taken great care to ensure the accuracy of our content, mistakes do occur. If you find a mistake in this book, either in the text or the code, we encourage you to report it to us. By doing so, you can save other readers from frustration and, of course, help us to improve the next version of this book. If you find any errata, please feel free to report them by visiting the following address:

<http://www.bouraspage.com>

Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, and added to any existing list of errata.

Conventions Used in This Book

Following are some explanations on the conventions used in this book. “Conventions” refer to the standard ways in which certain parts of the text are displayed.

PHP Statements

This book uses plenty of examples written in PHP language. PHP statements are shown in a typeface that looks like this.

```
This is a PHP statement
```

Keywords, Variables, Functions, and Arguments Within the Text of a Paragraph

Keywords, variables, functions, and arguments are sometimes shown within the text of a paragraph. When they are, the special text is shown in a typeface different from that of the rest of the paragraph. For instance, `$first_name = 5` is an example of a PHP statement within the paragraph text.

Words in Italics

You may notice that some of the special text is also displayed in italics. In this book, italicized words are general types that must be replaced with the specific name appropriate for your data. For example, the general form of a PHP statement may be presented as

```
echo arg1, arg2, arg3;
```

In order to complete the statement, the keywords *arg1*, *arg2*, and *arg3* must be replaced with something meaningful. When you use this statement in your program, you might use it in the following form.

```
echo $a, $b, "Hello";
```

Three dots (...): an Ellipsis

In the general form of a statement you may also notice three dots (...), also known as an “ellipsis,” following a list in an example. They are not part of the statement. An ellipsis indicates that you can have as many items in the list as you want. For example, the ellipsis in the general form of the `echo` statement

```
echo arg1, arg2, ...;
```

indicates that the list may contain more than two arguments. When you use this statement in your program, your statement might be something like this.

```
echo $a, $b, $c, $d, $e;
```

Square Brackets

The general form of some statements or functions may contain “square brackets” [], which indicate that the enclosed section is optional. For example, the general form of the statement

```
substr ( string , start [, length ] )
```

indicates that the section [, *length*] can be omitted.

The following two statements produce different results but they are both syntactically correct.

```
$a = substr ($s, $start);
$b = substr ($s, $start, $length);
```

The Dark Header

Most of this book's examples are shown in a typeface that looks like this.

file_31.2-3.php

```
<?php
    $a = 1;
    $b = 2;
    echo $a + $b;
?>
```

The dark header **file_31.2-3.php** on top indicates the filename that you must open to test the script. All the examples that contain this header can be found free of charge on the Internet. You can download them from the following address

<http://www.bouraspage.com>

Notices

Very often this book uses notices to help you better understand the meaning of a concept. Notices look like this.

Notice: *This typeface designates a note.*

Something Already Known or Something to Remember

Very often this book can help you recall something you have already learned (probably in a previous chapter). Other times, it will draw your attention to something you should memorize. Reminders look like this.

Remember! *This typeface designates something to recall or something that you should memorize.*

Section 1

Introductory Knowledge

Chapter 1

How a Computer Works

1.1 Introduction

In today's society, almost every task requires the use of a computer. In schools, students use computers to search the Internet and to send emails. At work, people use them to make presentations, to analyze data, and to communicate with customers. At home, people use them to play games and to chat with other people all over the world. Of course, don't forget smartphones such as iPhones. They are computers as well!

Computers can perform so many different tasks because of their ability to be programmed. In other words, a computer can perform any job that a program tells it to. A program is a set of statements (often called instructions or "commands") that a computer follows in order to perform a specific task.

Programs (usually referred as "application software") are essential to a computer, because without them a computer is a dummy machine that can do nothing at all. The program actually tells the computer what to do and when to do it. The programmer or the software developer is the person who designs, creates, and tests computer programs.

This book introduces you to the basic concepts of computer programming using the PHP language.

1.2 What is Hardware?

The term "hardware" refers to all devices or components that make up a computer. If you have ever opened the case of a computer or a laptop you have probably seen many of its components, such as the microprocessor (CPU), the memory, and the hard disk. A computer is not a device but a system of devices that all work together. The basic components of a typical computer system are discussed here.

➤ **The Central Processing Unit (CPU)**

This is the part of a computer that actually performs all the tasks defined in a program.

➤ **Main Memory (RAM – Random Access Memory)**

This is the area where the computer holds the program (while it is being executed/run) as well as the data that the program is working with. All programs and data stored in this type of memory are lost when you shut down your computer or you unplug it from the wall outlet.

➤ **Secondary Storage Devices**

This is usually the hard disk, and sometimes (but more rarely) the CD/DVD drive. In contrast to main memory, this type of memory can hold data for a longer period of time, even if there is no power to the computer. However, programs stored in this memory cannot be directly executed. They must be transferred to a much faster memory; that is, the main memory.

➤ **Input Devices**

Input devices are all those devices that collect data from the outside world and enter them into the computer for further processing. Keyboards, mice, and microphones are all input devices.

➤ Output Devices

Output devices are all those devices that output data to the outside world. Monitors (screens) and printers are output devices.

1.3 What is Software?

Everything that a computer does is under the control of software. There are two categories of software: system software and application software.

System software is the program that controls and manages the basic operations. For example, it controls the internal operations of a computer, manages all devices connected to it, saves data, loads data, and allows other programs to be executed. Windows, Linux, Mac OS X, Android, and iOS are all examples of system software. Another term for this category of programs is “operating systems.”

Application software refers to all the other programs that you use for your everyday tasks, such as browsers, word processors, notepads, games, and many more.

1.4 How a Computer Executes (Runs) a Program

When you turn on your computer, the main memory (RAM) is completely empty. The first thing the computer needs to do is to transfer the operating system from the hard disk to the main memory.

After the operating system is loaded to main memory, you can execute (run) any program (application software) you like. This is usually done by clicking, double clicking, or tapping the program’s corresponding icon. For example, let’s say you click on the icon of your favorite word processor. This action orders your computer to load (or copy) the word processing program from your hard disk to the main memory so the CPU can execute it.

Remember! *Programs are stored on secondary storage devices such as hard disks. When you install a program on your computer, the program is actually copied to your hard disk. But when you execute a program, the program is copied (loaded) from your hard disk to the main memory, and that copy of the program is executed.*

Notice: *The terms “run” and “execute” are synonymous.*

1.5 Compilers and Interpreters

Computers can execute programs that are written in a strictly defined computer language. You cannot write a program using a natural language such as English or Greek, because your computer won’t understand you!

But what does a computer actually understand? A computer can understand a specific low-level language called the “machine language.” In a machine language all statements (or commands) are made up of zeros and ones. The following is an example of a program written in a machine language, that calculates the sum of two numbers.

```
0010 0001 0000 0100
0001 0001 0000 0101
0011 0001 0000 0110
```

0111 0000 0000 0001

Shocked? Don't worry, you are not going to write programs this way. Hopefully, no one writes computer programs this way anymore. Nowadays, all programmers write their programs in a high-level language and then they use a special program to translate them into a machine language. There are two types of programs that programmers use to perform translation: compilers and interpreters.

A compiler is a program that translates statements written in a high-level language into a separate machine language program. The machine language program can then be executed any time you wish. After the translation, the compiler is no longer required.

An interpreter is a program that simultaneously translates and executes the statements written in a high-level language. As the interpreter reads each individual statement in the program, it translates it into a machine language code and then directly executes it. This process is repeated for every statement in the program. PHP is an example of a high-level language that usually uses an interpreter.

1.6 What is Source Code?

The statements (often called instructions or commands) that the programmer writes in a high-level language are called "source code" or simply "code." The programmer first types the source code into a program known as a code editor, and then uses either a compiler to translate it into a machine language program, or an interpreter to translate and execute it at the same time. NetBeans is an example of an Integrated Development Environment (IDE) that enables programmers to both write and execute their source code. You will learn more about NetBeans in Chapter 3.

1.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Modern computers can perform so many different tasks because they have many gigabytes of RAM.
2. A computer can operate without a program.
3. A hard disk is an example of hardware.
4. Data can be stored in main memory (RAM) for a long period of time, even if there is no power to the computer.
5. Data is stored in main memory (RAM), but programs are not.
6. Speakers are an example of an output device.
7. Windows and Linux are examples of software.
8. A media player is an example of system software.
9. When you turn on your computer, the main memory (RAM) already contains the operating system.
10. When you open your word processing application, it is actually copied from a secondary storage device to the main memory (RAM).
11. In a machine language, all statements (commands) are a sequence of zeros and ones.

12. Nowadays, a computer cannot understand zeros and ones.
13. Nowadays, software is written in a language composed of ones and zeros.
14. Software refers to the physical components of a computer.
15. In a high-level computer programming language, the computer does not understand zeros and ones.
16. The compiler and the interpreter are software.
17. The compiler translates source code to an executable file.
18. The interpreter creates a machine language program.
19. After the translation, the interpreter is not required anymore.
20. Source code can be written using a simple text editor.
21. Source code can be executed by a computer without compilation or interpretation.
22. A program written in machine language requires compilation (translation).
23. A compiler translates a program written in a high-level language.

1.8 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following is **not** computer hardware?
 - a. a hard disk
 - b. a DVD disc
 - c. a sound card
 - d. the main memory (RAM)
2. Which of the following is **not** a secondary storage device?
 - a. a DVD reader/writer device
 - b. a hard disk
 - c. a USB flash drive
 - d. RAM
3. Which one of the following operations **cannot** be performed by the CPU?
 - a. Transfer data to the main memory (RAM).
 - b. Display data to the user.
 - c. Transfer data from the main memory (RAM).
 - d. Perform arithmetic operations.
4. A touch screen is
 - a. an input device.
 - b. an output device.
 - c. both of the above

5. Which of the following is **not** software?
 - a. Windows
 - b. Linux
 - c. iOS
 - d. a video game
 - e. a web browser
 - f. All of the above are software.
6. Which of the following statements is correct?
 - a. Programs are stored on the hard disk.
 - b. Programs are stored on DVD discs.
 - c. Programs are stored in main memory (RAM).
 - d. All of the above are correct.
7. Which of the following statements is correct?
 - a. Programs can be executed directly from the hard disk.
 - b. Programs can be executed directly from a DVD disc.
 - c. Programs can be executed directly from the main memory (RAM).
 - d. All of the above are correct.
 - e. None of the above is correct.
8. Programmers **cannot** write computer programs in
 - a. PHP.
 - b. natural language such as English, Greek, and so on.
 - c. machine language.
9. A compiler translates
 - a. a program written in machine language into a high-level language program.
 - b. a program written in a natural language (English, Greek, etc.) into a machine language program.
 - c. a program written in high-level language into a machine language program.
 - d. none of the above
 - e. all of the above
10. Machine language is
 - a. a language that machines use to communicate with each other.
 - b. a language made up of numerical instructions that is used directly by a computer.
 - c. a language that uses English words for operations.

11. If two identical statements are one after the other, the interpreter
 - a. translates the first one and executes it, then it translates the second one and executes it.
 - b. translates the first one, then translates the second one, and then executes them both.
 - c. translates only the first one (since they are identical) and then executes it two times.

Chapter 2

PHP

2.1 What is PHP?

PHP is a widely used open source server-side scripting language that is especially suited for web development and can be combined with HTML. “Server side” means that scripts written in PHP are executed by a web server, such as Apache or IIS, with a PHP module installed.

While the acronym PHP originally stood for Personal Home Page, it now stands for “PHP: Hypertext Preprocessor.”

2.2 What is the Difference Between a Script and a Program?

Technically speaking, a script is *interpreted* whereas a program is *compiled*, but this is actually not their main difference. There is another small yet more important difference between them!

The main purpose of a script written in a scripting language such as PHP, JavaScript, or VBA (Visual Basic for Applications) is to control another application. So you can say that, in some ways, PHP controls the web server, JavaScript controls the web browser, and VBA controls a Microsoft Office application such as MS Word or MS Excel.

On the other hand, a program written in a programming language such as Java, C++, or Visual Basic executes independently of any other application. A program is compiled into a separate set of machine language instructions that can then be executed as stand-alone any time the user wishes.

Notice: *Macros of Microsoft Office are scripts written in VBA. Their purpose is to automate certain functions within Microsoft Office.*

Remember! *A script requires a hosting application in order to execute. A script cannot be executed as stand-alone.*

2.3 Why You Should Learn PHP

PHP is a very powerful server-side scripting language that is available on even the cheapest shared hosting servers. Some of the biggest websites on the Internet, including Facebook and Wikipedia, are built with PHP. Its coding style is quite easy to understand and it is very efficient on multiple platforms such as Windows, Linux, and Unix. PHP is a very flexible yet powerful language, making it most suitable for developing dynamic web pages.

One of PHP’s capabilities is the ability to interact with the file system of a computer. PHP can create files, write into files, or read the contents from files. It can also create directories, copy files, delete files, rename files, or even change file attributes. Even though PHP was originally designed to work in web pages, since it can perform almost any task related to the file system, it can be used for system administration tasks as well. For example, you can create a PHP script to back up your files or a script that processes text files by reformatting their contents.

Moreover, PHP can execute system commands or any other program installed on your system. Thus, programs written and compiled in C, C++, Perl, or any other computer language can be executed from PHP and PHP can make use of their output.

This enables you to add functionality to your scripts without spending time rewriting your old programs.

2.4 How PHP Works

Computers do not understand natural languages such as English or Greek, so you need a computer language such as PHP to communicate with them. PHP is what is known as a “high-level” computer scripting language. The PHP interpreter converts PHP language to a language that computers can actually understand, and that is known as the “machine language.”

The PHP interpreter comes in two flavors—the one that can be used with websites, and the other, which you can run from the command line (CLI: Command Line Interface). The latter is the one that this book actually uses.

In Chapter 3 you will learn how to install two software applications that you will need, XAMPP and NetBeans. XAMPP is actually a package of applications that includes a web server, an ftp server, and a mail server. However, the only component that you will need from this package is the CLI version of the PHP interpreter. NetBeans, on the other hand, is an Integrated Development Environment (IDE), the tool that can help you write PHP scripts.

Remember! *You must install both XAMPP and NetBeans. You need NetBeans in order to write PHP scripts, and you need the CLI version of the PHP interpreter (included in XAMPP) in order to execute them*

Chapter 3

Software Packages to Install

3.1 XAMPP

XAMPP is a free-of-charge package of open source applications. It includes a web server, an ftp server, a mail server, and the CLI version of the PHP interpreter that this book requires. Of course, XAMPP can be installed on different platforms such as Windows, Linux, and Mac OS X.

The acronym XAMPP stands for

- **X** (pronounced “cross”) – This means “cross-platform,” because the package can be operated on different platforms
- **A**pache – This is a web server
- **M**ySQL – This is a database server
- **P**HP – This is a programming language
- **P**erl – This is a programming language

3.2 How to Set Up XAMPP

In order to install XAMPP, you must download it free of charge from the following address:

<http://www.apachefriends.org>

Choose the version that is suitable for your platform. This book shows the process of installing XAMPP on a Windows platform. When the download completes, run the setup and when you are prompted to select the components you want to install, you can keep and install all the proposed components or you can check only those proposed in **Figure 3-1**.

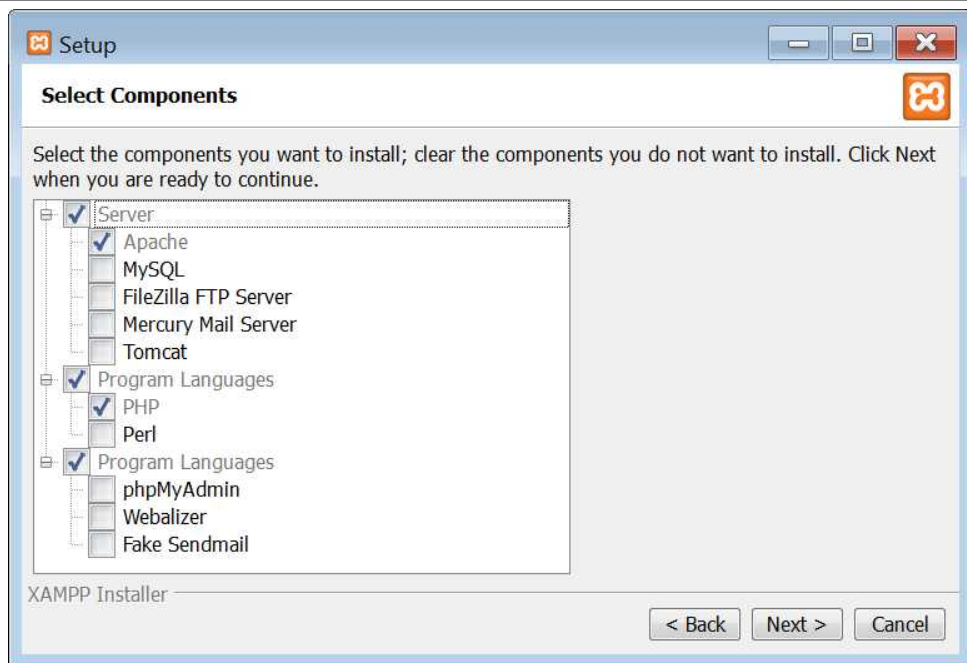


Figure 3-1 Selecting the components

Notice: There is an incompatibility issue between XAMPP and NetBeans IDE. You can find more information at the following address

<http://www.bouraspage.com>

When you are prompted to select the destination folder, it is advisable to leave the proposed “c:\xampp” as shown in **Figure 3-2**. Especially for Windows Vista, Windows 7, and Windows 8, problems may arise due to the lack of writing permissions on folders other than the one suggested.

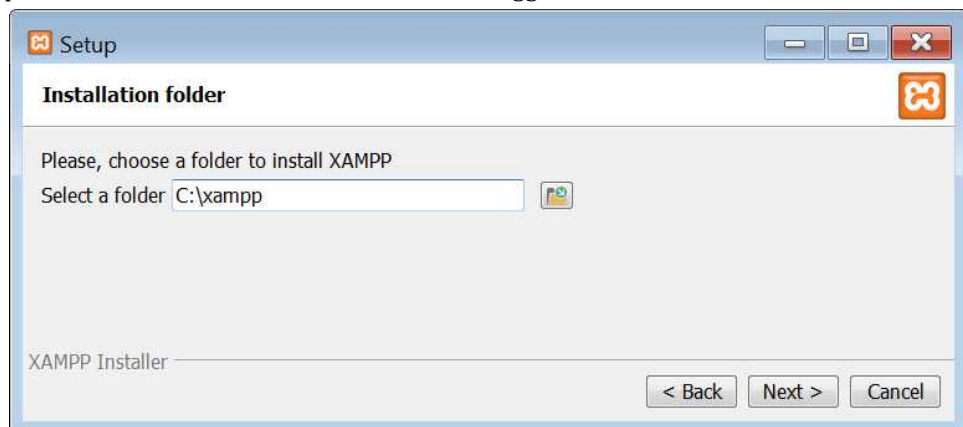


Figure 3-2 Selecting the destination folder

When the installation process is complete, you will need to make some modifications to the PHP configuration file. Using the Windows Notepad application or any equivalent text editor (for example, Notepad++), edit the “C:\xampp\php\php.ini” file. Find the [XDebug] section (usually at the end of the file) and uncomment some lines by removing the semicolon (;) from the beginning of the lines as shown here.

```
[XDebug]
zend_extension = "C:\xampp\php\ext\php_xdebug.dll"
;xdebug.profiler_append = 0
;xdebug.profiler_enable = 1
;xdebug.profiler_enable_trigger = 0
;xdebug.profiler_output_dir = "C:\xampp\tmp"
;xdebug.profiler_output_name = "cachegrind.out.%t-%s"
xdebug.remote_enable = 1
xdebug.remote_handler = "dbgp"
xdebug.remote_host = "127.0.0.1"
xdebug.trace_output_dir = "C:\xampp\tmp"
```

Save your changes and close the text editor.

Notice: If [XDebug] section is not present in your php.ini file, just add it manually.

Notice: If php_xdebug.dll file is not present in your C:\xampp\php\ext\ folder, download it free of charge from the following address:

<https://xdebug.org/download.php>

Your PHP interpreter is now ready to use. However, one last thing has to be done. You must install NetBeans and configure it so you can use the CLI version of the PHP interpreter. Once you do this, you will be able to write your PHP scripts and execute them directly from NetBeans.

3.3 NetBeans

NetBeans is an Integrated Development Environment (IDE) that provides a great set of tools for PHP and lets you easily write PHP scripts. NetBeans is much more than a text editor. It can indent lines, match words and brackets, and highlight source code that is written incorrectly. It also provides automatic code, which means that as you type, it displays a list of possible completions.

The IDE also provides hints to help you analyze your code and find any potential problems. It even suggests some simple solutions to fix those problems.

You can use the NetBeans IDE not only to write but also to execute your scripts directly from the IDE, thanks to the CLI version of the PHP interpreter discussed in Chapter 2.

NetBeans IDE can be installed on all operating systems that support Java, from Windows to Linux to Mac OS X systems. NetBeans is free of charge and open source, and it has a large community of users and developers all around the world.

3.4 How to Set Up NetBeans IDE

In order to install NetBeans IDE, you must download it, free of charge, from the following address:

<https://netbeans.org/downloads/index.html>

On the NetBeans web page, there is a dropdown list that lets you choose the required platform (Windows, Linux, or Mac OS X). Choose the platform that matches yours. This book shows the process of installing NetBeans IDE on a Windows platform.

From all the “download bundles” available on netbeans.org, select and download the one called “All.” When the download is complete, run the corresponding installer.

If you receive the message shown in **Figure 3-3**, it means that the Java SE Development Kit (JDK) was not found on your system. Please visit the displayed address to download and install the JDK before proceeding with the NetBeans IDE installation.

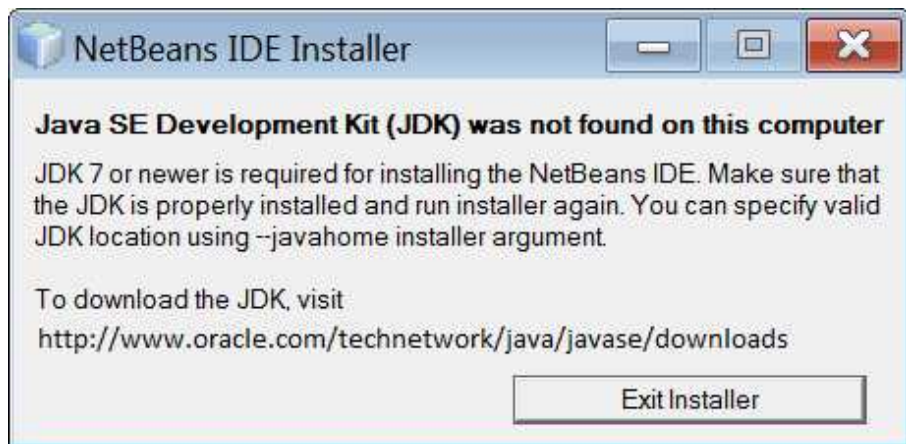


Figure 3-3 This message is displayed when the JDK is not found on your system.

On the first screen of the NetBeans IDE installer, you can keep and install all the proposed packs and runtimes or you can click on the “Customize” button and check only those proposed in **Figure 3-4**.

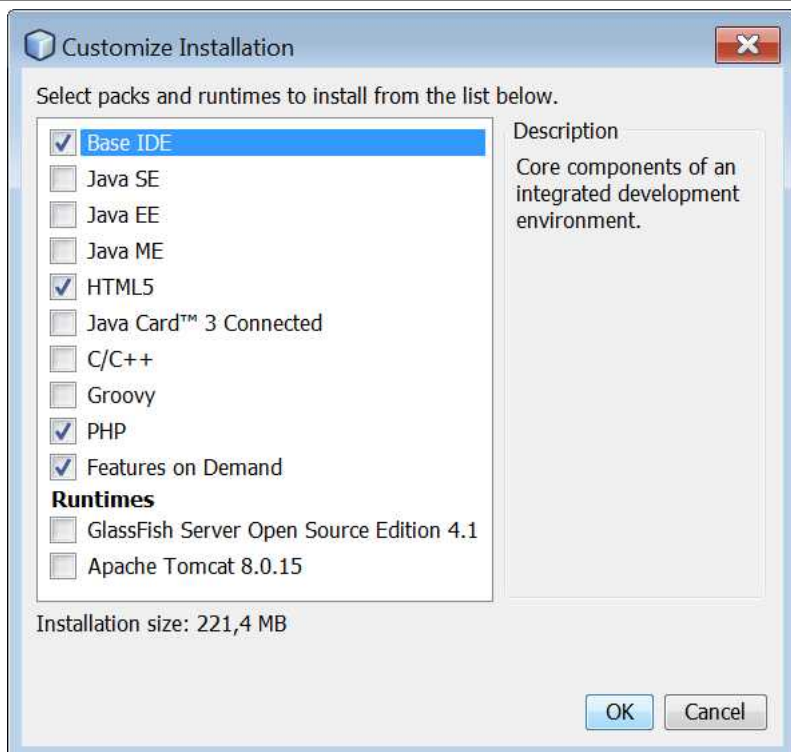


Figure 3-4 Selecting the packs and runtimes to install in the NetBeans IDE installer

Click on the “Next” button to go through the installation screens. When the license agreement shows up, you must read and accept the terms. The next screen prompts you to select the installation folder and the Java Environment. Leave the proposed folders as shown in **Figure 3-5**.



Figure 3-5 Selecting the installation folders in the NetBeans IDE installer

Notice: The installation folder and the Java environment proposed on your computer may differ from those in **Figure 3-5** depending on the versions of the NetBeans IDE or JDK that you have.

On the next screen, leave the field “Check for Updates” checked, as shown in **Figure 3-6**. This option will automatically check for updates of installed plugins.



Figure 3-6 Selecting the box to check for updates

Click on the “Install” button.

When the installation process is complete, you will need to change some settings in order to let PHP scripts run directly from the NetBeans IDE.

Open the NetBeans IDE. The first screen that you will see is shown in **Figure 3-7**.

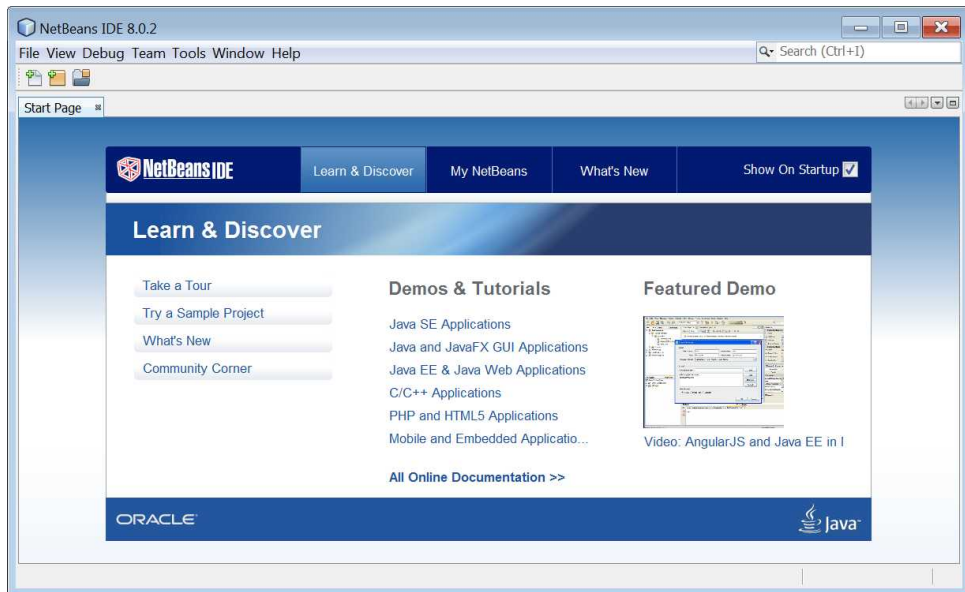


Figure 3-7 The NetBeans IDE start page

From the main menu select Tools→Options. In the popup window that appears, click on the PHP icon to open the PHP options dialog box, as shown in **Figure 3-8**.

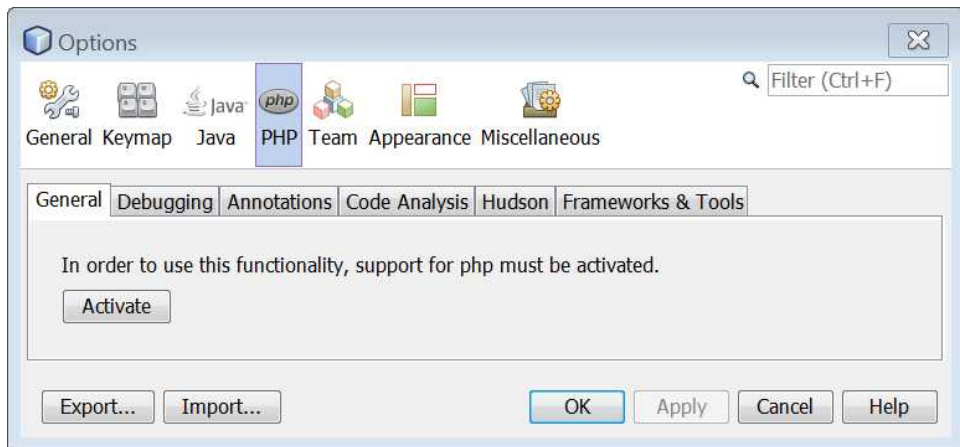


Figure 3-8 The PHP Options dialog box

In order to use this functionality, you must activate support for PHP. Click on the “Activate” button. The window now becomes as shown in **Figure 3-9**.

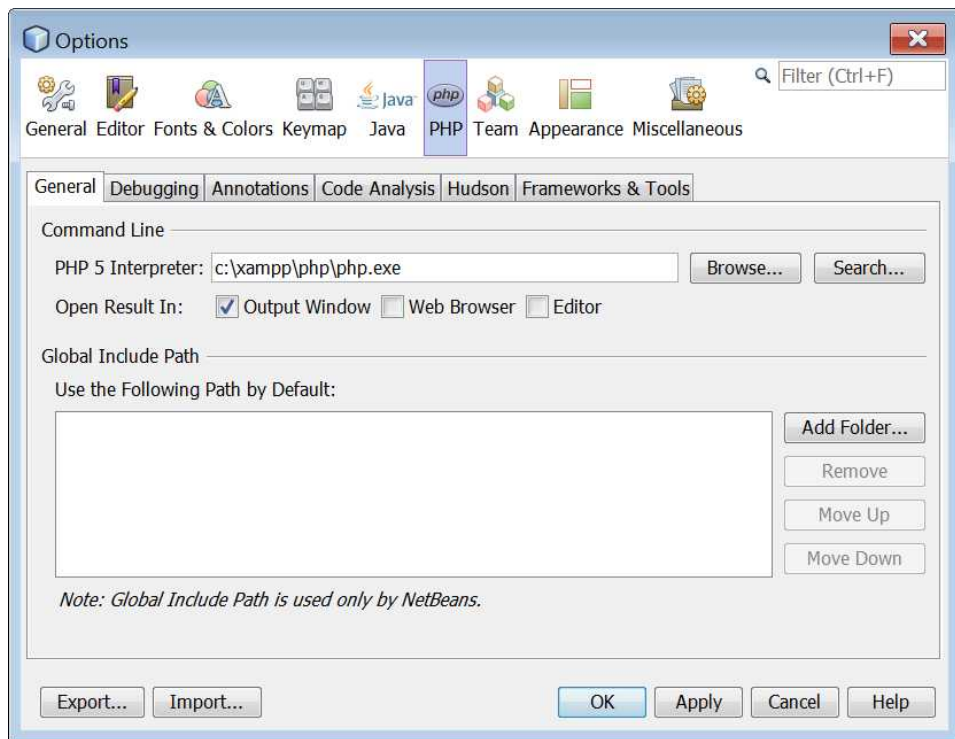


Figure 3-9 The PHP Options dialog box after PHP activation

In the “PHP 5 Interpreter” field, specify the path to the “php.exe” file. Use the “Browse...” or “Search...” button, if necessary. If you haven’t already changed XAMPP’s folder during the installation process, this path should be “c:\xampp\php\php.exe.”

One additional thing that this book does in order to save paper is to decrease the number of spaces per indent to two. You can also change this setting by first clicking on the Editor icon to open the Editor’s “Options” dialog box, as shown in **Figure 3-10**.

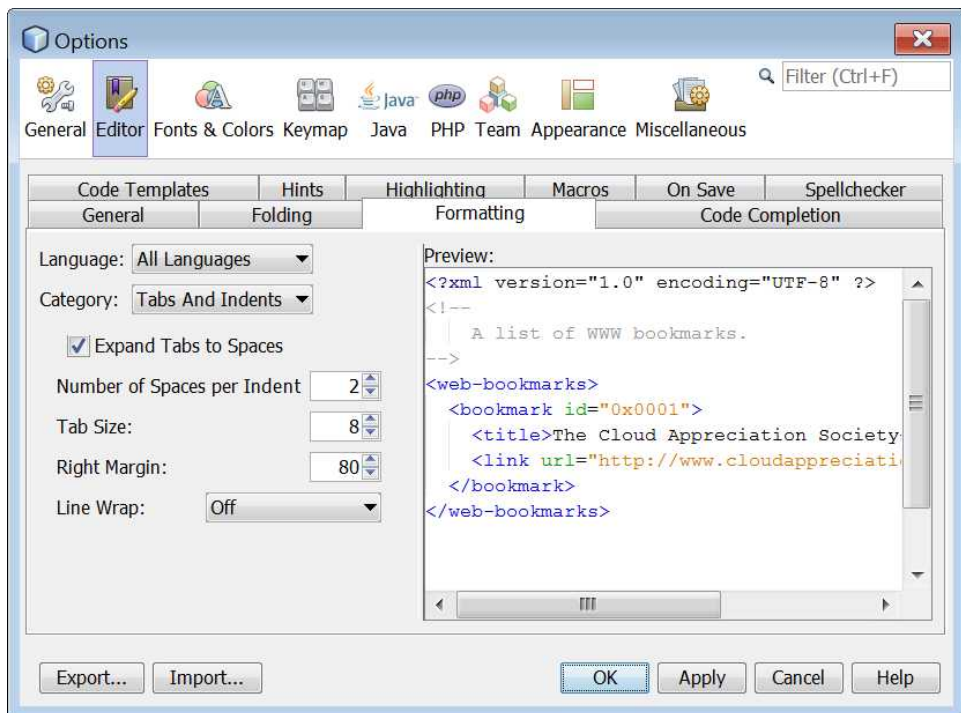


Figure 3–10 The Editor’s “Options” dialog box

On the “Formatting” tab, set the field “Number of Spaces per Indent” to 2.

Click on the “OK” button.

NetBeans IDE has been configured properly! Now it’s time to conquer the world of PHP! In the upcoming chapters you will learn all about how to create PHP projects, how to write PHP scripts, and so many tips and tricks useful in your first steps as a budding programmer!

Review Questions in “Introductory Knowledge”

Answer the following questions.

1. What is hardware?
2. List the five basic components of a typical computer system.
3. Which part of the computer actually executes the programs?
4. Which part of the computer holds the program and its data while the program is running?
5. Which part of the computer holds data for a long period of time, even when there is no power to the computer?
6. How do you call the device that collects data from the outside world and enters them into the computer?
7. List some examples of input devices.
8. How do you call the device that outputs data from the computer to the outside world?
9. List some examples of output devices.
10. What is software?
11. How many software categories are there, and what are their names?
12. A word processing program belongs to what category of software?
13. What is a compiler?
14. What is an interpreter?
15. What is meant by the term “machine language”?
16. What is source code?
17. What is PHP?
18. What is the difference between a script and a program?
19. What does the acronym PHP now stand for?
20. What are some of the possible uses of PHP?
21. What does the acronym CLI stand for?
22. What is XAMPP?
23. What does the acronym XAMPP stand for?
24. What is NetBeans IDE?

Section 2

Getting Started with PHP

Chapter 4

Introduction to Basic Algorithmic Concepts

4.1 What is an Algorithm?

An algorithm is a strictly defined finite sequence of well-defined statements (often called instructions or commands) that provides the solution to a problem or to a specific class of problems for any acceptable set of input values (if there are any inputs). In other words, an algorithm is a step-by-step procedure to solve a given problem. The term “finite” means that the algorithm should reach an end point and cannot run forever.

You can find algorithms everywhere in your real life, not just in computer science. For example, the process for preparing toast or a cup of tea can be expressed as an algorithm. Certain steps, in a certain order, must be followed in order to achieve your goal.

4.2 The Algorithm for Making a Cup of Tea

The following is an algorithm for making a cup of tea.

1. Put the teabag in a cup.
2. Fill the kettle with water.
3. Boil the water in the kettle.
4. Pour some of the boiled water into the cup.
5. Add milk to the cup.
6. Add sugar to the cup.
7. Stir the tea.
8. Drink the tea.

As you can see, there are certain steps that must be followed. These steps are in a specific order, even though some of the steps could be rearranged. For example, steps 5 and 6 can be reversed. You could add the sugar first, and the milk afterwards.

Please keep in mind that the order of some steps can probably be changed but you can't move them far away from where they should be. For example, you can't move step 3 (“Boil the water in the kettle.”) to the end of the algorithm, because you will end up drinking a cup of iced tea (and not a warm one) which is totally different from your initial goal!

4.3 Properties of an Algorithm

An algorithm must satisfy the following properties:

- **Input:** The algorithm must have input values from a specified set.
- **Output:** The algorithm must produce the output values from a specified set of input values. The output values are the solution to a problem.
- **Finiteness:** For any input, the algorithm must terminate after a finite number of steps.
- **Definiteness:** All steps of the algorithm must be precisely defined.

- **Effectiveness:** It must be possible to perform each step of the algorithm correctly and in a finite amount of time. That is, its steps must be basic enough so that, for example, someone using a pencil and a paper could carry them out exactly, and in a finite amount of time. It is not enough that each step is definite (or precisely defined), but it must also be feasible.

4.4 Okay About Algorithms. But What Is a Computer Program Anyway?

A computer program is nothing more than an algorithm that is written in a language that computers can understand, like PHP, C++, C#, or Java.

A computer program cannot actually *make* you a cup of tea or cook your dinner, although an algorithm can guide you through the steps to do it yourself. However, programs can (for example) be used to calculate the average value of a set of numbers, or to find the maximum value among them. Artificial intelligence programs can even play chess or solve logic puzzles.

Notice: Keep in mind that the term “PHP program” is not used very often. The most commonly used term is “PHP script,” and that is the term this book will use from now on.

4.5 The Party of Three!

There are always three parties involved in an algorithm—the one that writes the algorithm, the one that executes it, and the one that uses or enjoys it.

Let’s take an algorithm for preparing a meal, for example. Someone writes the algorithm (the author of the recipe book), someone executes it (probably your mother, who prepared the meal following the steps from the recipe book), and someone uses it (probably you, who enjoys the meal).

Now consider a real computer program. Let’s take a video game, for example. Someone writes the algorithm in a computer language (the programmer), someone or something executes it (usually a laptop or a computer), and someone else uses it or plays with it (the user).

Be careful, because sometimes the terms “programmer” and “user” can be confusing. When you *write* a computer program, for that period of time you are “the programmer.” However, when you *use* your own program, you are “the user.”

4.6 The Three Main Stages Involved in Creating an Algorithm

Three main stages are involved in creating an algorithm: data input, data processing, and results output. The order is specific and cannot be changed.

Consider a computer program that finds the average value of three numbers. First, the program must prompt the user to enter the numbers (data input). Next, the program calculates the average value of the numbers (data processing). Finally, the program displays the result on the computer’s screen (results output).

Let's take a look at these stages in more detail.

1. Prompt the user to enter a number.

2. Prompt the user to enter a second number.

3. Prompt the user to enter a third number.

}

Input

4. Calculate the sum of the three numbers.

5. Divide the sum by 3.

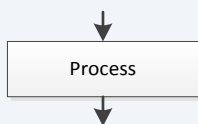
}

Processing

6. Display the result on the screen.

}

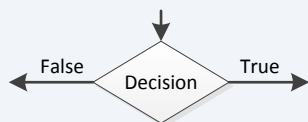
Output
- In some rare situations, the input stage may be absent and the computer program may be composed of only two stages. For example, consider a computer program that is written to calculate the following sum.
- $$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10$$
- In this example, the user enters no values at all because the computer knows exactly what to do. It must calculate the sum of the numbers 1 to 10.
- Now let's take a look at the same example, slightly altered. Consider a computer program that is written to calculate the following sum.
- $$1 + 2 + 3 + \cdots + N$$
- Of course, this sum is not the same as the previous one. In this example, the user needs to input some data. The computer cannot decide by itself about the exact value of number N. This value must be entered by the user. Once the user enters a value for N, the computer can proceed to calculate the result. For example, if the user enters the number 5, the computer can then find the result of $1 + 2 + 3 + 4 + 5$.
- ### 4.7 Flowcharts
- A flowchart is a graphical method of presenting an algorithm, usually on paper. It is the visual representation of the algorithm's flow of execution. In other words, it visually represents how the flow of execution proceeds from one statement to the next until the end of the algorithm is reached. A flowchart cannot be entered directly into a computer as is. It must first be converted into a programming language such as PHP.
- The basic symbols that flowcharts use are shown in **Table 4-1**.
- Table 4-1** Flowchart Symbols and Their Functions
- | | |
|---|---|
|  | Start/End: Represents the beginning or the end of an algorithm. The Start symbol has one exit and the End symbol has one entrance. |
|  | Arrow: Shows the flow of execution. An arrow coming from one symbol and ending at another symbol shows that control passes to the symbol that the arrow is pointing to. Arrows are always drawn as straight lines going up and down or sideways (never at an angle). |



Process: Represents a process or mathematical (formula) calculation. The Process symbol has one entrance and one exit.



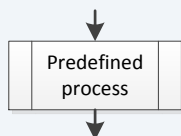
Data Input/Output: Represents the data input or the results output. In most cases, data comes from a keyboard and results are displayed on a screen. The Data input/output symbol has one entrance and one exit.



Decision: Indicates the point at which a decision is made. Based on a given condition (which can be true or false), the algorithm will follow either the right or the left path. The Decision symbol has one entrance and two (and always only two) exits.



Off-page connectors: Show continuation of a flowchart onto another page. They are used to connect segments on multiple pages when a flowchart gets too big to fit onto one sheet of paper. The outgoing off-page connector symbol has one entrance and the incoming off-page connector symbol has one exit.



Subprogram (predefined process): Depicts subprograms that are formally defined elsewhere, such as in a separate flowchart. The Predefined process symbol has one entrance and one exit.

An example of a flowchart is shown in **Figure 4-1**. The algorithm prompts the user to enter three numbers and then calculates their average value and displays it on the computer screen.

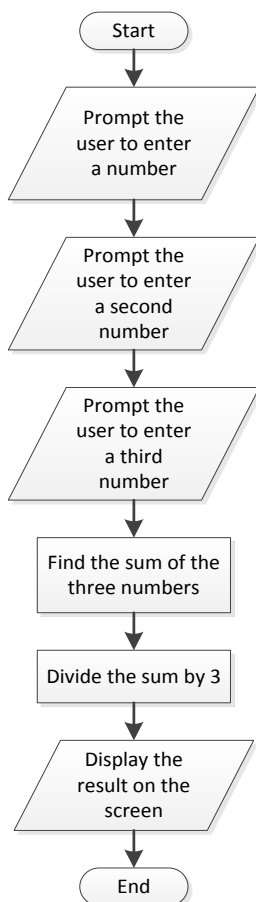


Figure 4–1 Flowchart for an algorithm that calculates and displays the average of three numbers

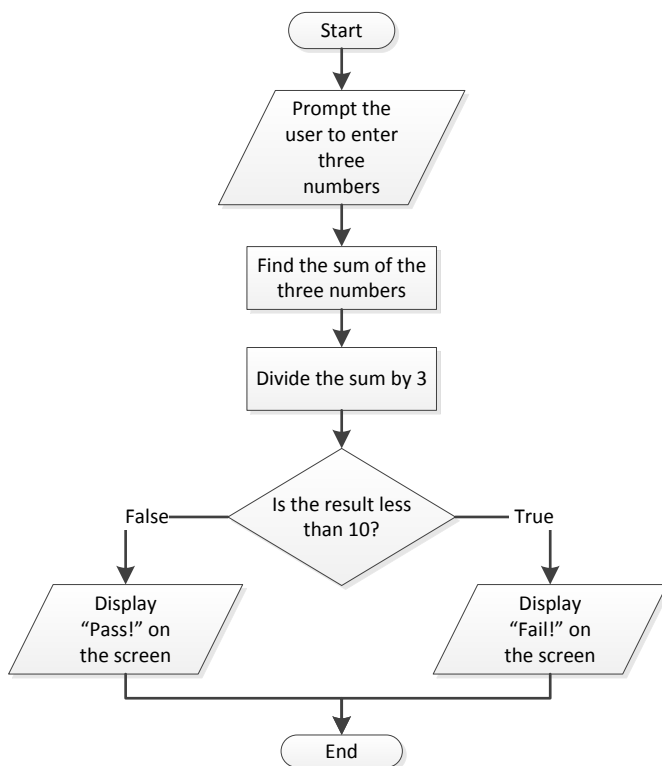
Remember! A flowchart always begins and ends with a Start/End symbol!

Exercise 4.7-1 Finding the Average Value of Three Numbers

Design an algorithm that calculates the average value of three numbers. Whenever the average value is below 10, a message “Fail!” should be displayed. Otherwise, if the average value is 10 or above, a message “Pass!” should be displayed.

Solution

In this problem, two different messages must be displayed, but only one can appear each time the algorithm is executed; the wording of the message depends on the average value. The flowchart for the algorithm is presented next.



Notice: To save paper, you can prompt the user to enter all three numbers using one single oblique parallelogram.

Remember! A Decision symbol always has one entrance and two exit paths!

Of course it is very soon for you to start creating your own algorithms. This particular exercise is quite simple and is presented in this chapter as an exception, just for demonstration purposes. You need to learn more before you start creating your own algorithms or even PHP scripts. Just be patient! In a few chapters the big moment will come!

4.8 What are "Reserved Words"?

In a computer language, "reserved words" are all those words that have a strictly predefined meaning—they are reserved for special use and cannot be used for any other purpose. For example, the words `Start`, `End`, `Read`, and `Write` in flowcharts have a predefined meaning. They are used to represent the beginning, the end, the data input, and the results output, respectively.

Reserved words exist in all high-level computer languages as well. In PHP, there are many reserved words such as `echo`, `if`, `while`, and `for`. Their meanings are predefined, so these words cannot be used for any other purposes.

Notice: Reserved words in a computer language are often called keywords.

4.9 What Is the Difference Between a Statement and a Command?

There is a big discussion on the Internet about whether there is, or is not, any difference between a statement and a command. Some people prefer to use the term “statement,” and some others the term “command.” For a novice programmer, there is no difference; both are instructions to the computer!

4.10 What Is Structured Programming?

Structured programming is a software development method that uses modularization and structured design. This means that large programs or scripts are broken down into smaller modules and each individual module uses structured code, which means that the statements are organized in a specific manner that minimizes errors and misinterpretation. As its name suggests, structured programming is done in a structured programming language and PHP is one such language.

The structured programming concept was formalized in 1966 by Corrado Böhm¹ and Giuseppe Jacopini². They demonstrated theoretical computer program design using sequences, decisions, and iterations.

4.11 The Three Fundamental Control Structures

There are three fundamental control structures in structured programming.

- **Sequence Control Structure:** This refers to the line-by-line execution, in which statements are executed sequentially, in the same order in which they appear in the script. They might, for example, carry out a series of read or write operations, arithmetic operations, or assignments to variables.
- **Decision Control Structure:** Depending on whether a condition is true or false, the decision control structure may skip the execution of an entire block of statements or even execute one block of statements instead of another.
- **Loop Control Structure:** This is a control structure that allows the execution of a block of statements multiple times until a specified condition is met.

If you didn't quite understand the deeper meaning of these three control structures, don't worry, because upcoming chapters will analyze them very thoroughly. Patience is a virtue. All you have to do for now is wait!

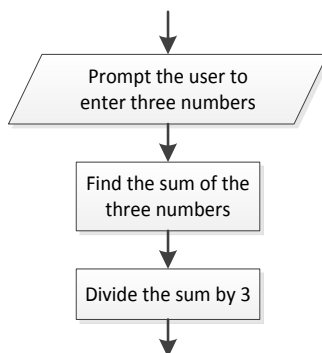
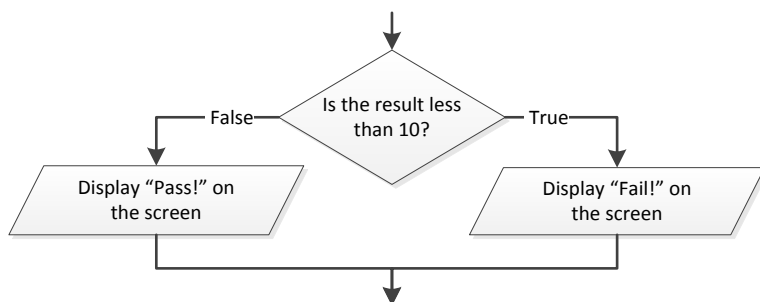
¹ Corrado Böhm (1923–) is a computer scientist known especially for his contribution to the theory of structured programming, and for the implementation of functional programming languages.

² Giuseppe Jacopini (1936–2001) was a computer scientist. His most influential contribution is the theorem about structured programming, published along with Corrado Böhm in 1966, under the title “Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules.”

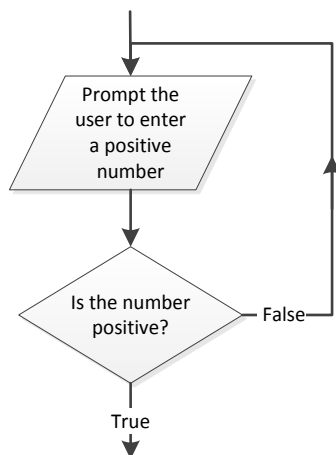
(Bohm, C., and Jacopini, G., “Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules,” *Communications of the ACM* 9(5), (May, 1966), 366–371.)

Exercise 4.11-1 *Understanding Control Structures Using Flowcharts*

Using flowcharts, give an example for each type of control structure.

Solution**Example of a Sequence Control Structure****Example of a Decision Control Structure**

Example of a Loop Control Structure



4.12 Your First PHP Script

Converting a flowchart to a computer language such as PHP results in a PHP script. A PHP script is nothing more than a text file including PHP statements. PHP scripts can even be written in your text editor application! Keep in mind, though, that using NetBeans IDE to write PHP scripts is a much better solution due to all of its included features that can make your life easier.

A PHP script always begins and ends with the delimiters `<?php` and `?>`, respectively, as shown here.

```
<?php
// PHP code goes here
?>
```

A PHP script is saved on your hard disk with the default `.php` file extension.

Here is a very simple PHP script that displays just three messages on the screen.

```
<?php
echo "Hello World!";
echo "Hallo Welt!";
echo "The End";
?>
```

Notice: Please note that PHP requires that all statements be terminated with a semicolon.

4.13 What is the Difference Between Syntax Errors and Logic Errors?

When programmers write code in a high-level language there are two types of errors that they might make: syntax errors and logic errors.

Syntax errors are mistakes such as misspelled keywords, a missing punctuation character, a missing bracket, or a missing closing parenthesis. Fortunately, NetBeans IDE detects these errors as you type and displays a red exclamation mark in front of the erroneous line. If you try to execute a PHP script that includes syntax errors, you will get error messages on your screen and the program won't be executed. You must correct all the errors and then try to execute the script again.

Logic errors are those errors that prevent your program from doing what you expected it to do. With logic errors you get no warning at all. Your code may compile and run but the result is not the expected one. Logic errors are the most difficult errors to detect. You must revisit your script thoroughly to determine where your error is. For example, consider a PHP script that prompts the user to enter three numbers, and then calculates and displays their average value. The programmer, however, made a typographical error; one of his or her statements divides the sum of the three numbers by 5, and not by 3 as it should. Of course the PHP script is executed as usual, without any error messages, prompting the user to enter three numbers and displaying a result, but obviously not the correct one! It is the programmer who has to find and correct the erroneously written PHP statement, not the computer or the compiler!

4.14 Commenting Your Code

When you write a small and easy script, anyone can understand how it works just by reading it line-by-line. However, long scripts are difficult to understand, sometimes even by the same person who wrote them. Comments are extra information that can be included in a script to make it easier to read and understand. You can add explanations and other pieces of information, including:

- who wrote the script
- when the script was created or last modified
- what the script does
- how the script works

However, you should not over-comment. There is no need to explain every line of your script. Add comments only when a specific portion of your script is hard to follow.

In PHP, you can add comments using one of the following methods:

- double slashes `//.....`
- the sharp symbol `#.....`
- slash-asterisk, asterisk-slash delimiters `/* ..... */`

The following script demonstrates how to use all types of commenting. Usually double slashes (`//`) and the sharp symbol (`#`) are used for commenting one single line, whereas the slash-asterisk, asterisk-slash delimiters `/* ..... */` are used for commenting multiple lines at once.

```
<?php
```

```
/*
```

```
Created By Bouras Aristides
```



```
Date created: 12/25/2003
Date modified: 04/03/2008
Description: This script displays some messages on the screen
*/

echo "Hello Zeus!"; //display a message on the screen

//display a second message on the screen
echo "Hello Hera!";

/* display a third message on screen */ echo "Τελα σου";

#This is a comment      echo "The End";

?>
```

As you can see in the preceding script, you can add comments above a statement or at the end of it, but not in front of it. Look at the last `echo` statement, which is supposed to display the message “The End.” This statement is never executed because it is considered part of the comment.

Notice: If you add comments using the delimiters `/* ..... */` in front of a statement, the statement is still executed. In the preceding example, the Greek message “Τελα σου Ποσειδώννα,” even though it is written next to some comments, is still executed. It is advisable, however, not to follow this writing style because it can make your code difficult to read.

4.15 User-Friendly Programs and Scripts

What is a “user-friendly” program or script? It’s the one that the user considers a friend instead of an enemy, the one that can be used easily by a novice user.

If you want to write user-friendly programs or scripts you have to put yourself in the shoes of the user. Users want the computer to do their job their way, with a minimum of effort. Hidden menus, imprecise labels and directions, and misleading error messages are all things that can make a program or script user-unfriendly!

The law that best defines user-friendly designs is the Law of Least Astonishment: “The program should act in a way that least astonishes the user.” This law is also commonly referred to as the Principle of Least Astonishment (POLA).

4.16 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The process for preparing a meal is actually an algorithm.
2. Algorithms are used only in computer science.
3. An algorithm can run forever.
4. In an algorithm, you can relocate a step in any position you wish.

5. An algorithm should produce the correct output values for just one set of input values.
6. Computers can play chess.
7. An algorithm can always become a computer program.
8. Programming is the process of creating a computer program.
9. There are always three parties involved in a computer program: the programmer, the computer, and the user.
10. The programmer and the user can sometimes be the same person.
11. It is possible for a computer program to output no results.
12. A flowchart is a computer program.
13. A flowchart is composed of a set of geometric shapes.
14. A flowchart is a method used to represent an algorithm.
15. You can design a flowchart without using any Start/End symbols.
16. You can design a flowchart without using any Process symbols.
17. You can design a flowchart without using any Data input/output symbols.
18. A flowchart should always include at least one Decision symbol.
19. In a flowchart, a Decision symbol can have one, two, or three exit paths, depending on the given problem.
20. Reserved words are all those words that have a strictly predefined meaning.
21. Structured programming includes structured design.
22. PHP is a structured computer language.
23. The basic principle of structured programming is that it includes only four fundamental control structures.
24. One statement, written ten times, is considered a loop control structure.
25. Decision control structure refers to the line-by-line execution.
26. A misspelled keyword is considered a logic error.
27. A PHP script can be executed even though it contains logic errors.
28. A missing semicolon at the end of a statement is considered a syntax error.
29. An exclamation mark at the end of a statement cannot prevent the whole PHP script from being executed.
30. One of the advantages of structured programming is that no errors are made while writing a computer program.
31. Logic errors are caught during compilation.
32. Syntax errors are the most difficult errors to detect.
33. A script that calculates the area of a triangle but outputs the wrong results contains logic errors.
34. When a script includes no output statements, it contains syntax errors.
35. A script should always contain comments.
36. If you add comments to a script, the computer can more easily understand it.

37. You can add comments anywhere in a script.
38. Comments are not visible to the users of a program.
39. A program or script is called user-friendly if it can be used easily by a novice user.
40. The acronym POLA stands for “Principle of Least Amusement.”

4.17 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. An algorithm is a strictly defined finite sequence of well-defined statements that provides the solution to
 - a. a problem.
 - b. a specific class of problems.
 - c. both of the above
2. Which of the following is **not** a property that an algorithm should satisfy?
 - a. effectiveness
 - b. fittingness
 - c. definiteness
 - d. input
3. A computer program is
 - a. an algorithm.
 - b. a sequence of instructions.
 - c. both of the above
 - d. none of the above
4. When someone prepares a meal, he or she is the
 - a. “programmer.”
 - b. “user.”
 - c. none of the above
5. Which of the following does **not** belong in the three main stages involved in creating an algorithm?
 - a. data production
 - b. data input
 - c. data output
 - d. data processing
6. A flowchart can be
 - a. presented on a piece of paper.
 - b. entered directly into a computer as is.
 - c. both of the above

7. A rectangle in a flowchart represents
 - a. input/output.
 - b. a processing operation.
 - c. a decision.
 - d. none of the above
8. Which of the following is/are control structures?
 - a. a decision
 - b. a sequence
 - c. a loop
 - d. All of the above are control structures.
9. Which of the following PHP statements contains a syntax error?
 - a. `echo "Hello Poseidon"`
 - b. `echo "It's me! I am the one that contains a syntax error!!!";`
 - c. `echo "Hello Athena";`
 - d. none of the above
10. Which of the following `echo` statements are actually executed?
 - a. `//echo "Hello Apollo";`
 - b. `/* echo "Hello Artemis"; */`
 - c. `//This will be executed // echo "Hello Ares";`
 - d. `/* This will be executed */ echo "Hello Aphrodite";`
 - e. none of the above

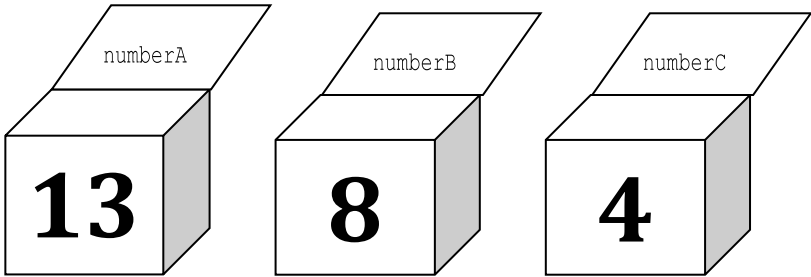
Chapter 5

Variables and Constants

5.1 What is a Variable?

In computer science, a variable is a location in the computer's main memory (RAM) where you can store a value and change it as the program executes.

Picture a variable as a transparent box in which you can insert and hold one thing at a time. Because the box is transparent, you can also examine what it contains. Moreover, if you have two or more boxes you can give each box a unique name. For example, you could have three boxes, each containing a different number, and you could name the boxes `numberA`, `numberB`, and `numberC`.

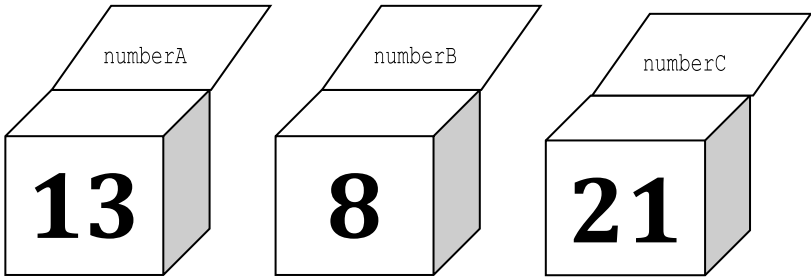


The boxes named `numberA`, `numberB` and `numberC` in the example contain the numbers 13, 8, and 4, respectively. Of course, you can examine or even alter the contained value of each one of these boxes at any time.

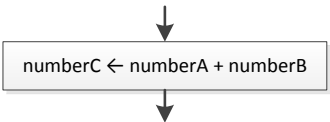
Now, let's say that someone asks you to find the sum of the values of the first two boxes and then store the result in the last box. The steps you must follow are:

1. Take a look at the first two boxes and examine the contained values.
2. Use your CPU (this is your brain) to calculate the sum (the result).
3. Insert the result in the last box. However, since each box can contain only one single value at a time, the value 4 is actually replaced by the number 21.

The boxes now look like this.



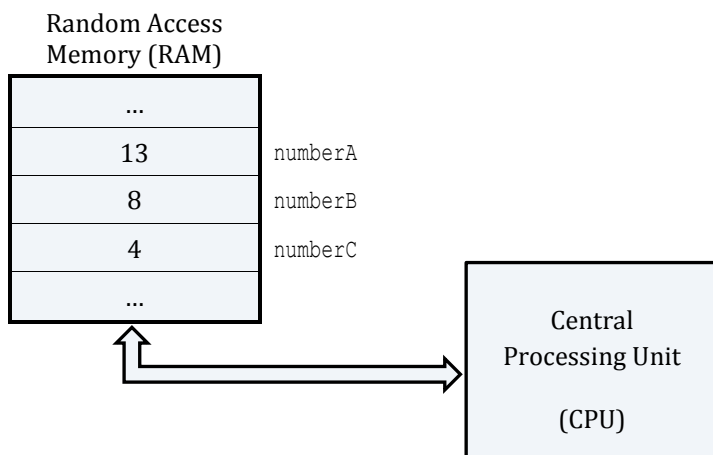
In a flowchart, the action of storing a value in a variable is represented by a left arrow



This action is usually expressed as “Assign a value, or the result of an expression, to a variable.” The left arrow is called the “value assignment operator.”

Notice: Please note that this arrow always points to the left. You are not allowed to use right arrows. Also, on the left side of the arrow only one single variable should exist.

In real computer science, the three boxes are actually three individual regions in main memory (RAM), named `numberA`, `numberB` and `numberC`.



When a program instructs the CPU to execute the statement

`numberC ← numberA + numberB`

it follows the same three-step process as in the previous example.

1. The number 13 is transferred from the RAM’s region named `numberA` to the CPU.

The number 8 is transferred from the RAM’s region named `numberB` to the CPU.

(This is the first step, in which you examined the values contained in the first two boxes.)

2. The CPU calculates the sum of $13 + 8$.

(This is the second step, in which you used your brain to calculate the sum, or result.)

3. The result, 21, is transferred from the CPU to the RAM’s region named `numberC`, replacing the existing number 4.

(This is the third step, in which you inserted the result in the last box.)

After execution, the RAM looks like this.

RAM	
...	
13	numberA
8	numberB
21	numberC
...	

Remember! While a PHP script is running a variable can hold various values, but only one value at a time. When you assign a value to a variable, this value remains stored until you assign a new value. The old value is lost.

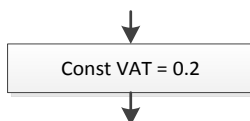
A variable is one of the most important elements in computer science because it helps you interact with data stored in the main memory (RAM). Soon, you will learn all about how to declare and use variables in PHP.

5.2 What is a Constant?

Sometimes you may need to use a value that cannot change while the program is running. Such a value is called a “constant.” In simple terms, you could say that a constant is a locked variable. This means that when a program begins to run and a value is assigned to the constant, nothing can change the value of the constant while the program is running. For example, in a financial program an interest rate can be declared as a constant.

A descriptive name for a constant can also improve the readability of your program and help you avoid errors. For example, let’s say that you are using the value 3.14159265 (but not as a constant) at many points throughout your program. If you make an error when typing the number, this will produce the wrong results. But, if this value is given a name, any typographical error in the name is detected by PHP, and you are notified with an error message.

In a flowchart, you can represent the action of setting a constant equal to a value with the equals sign, (=).



Notice: This book uses the reserved word `Const` to distinguish a constant from a variable.

Consider an algorithm that lets the user enter the prices of three different products and then calculates and displays the 20% Value Added Tax (known as VAT) for each product. The flowchart in **Figure 5-1** shows this process when no constant is used.

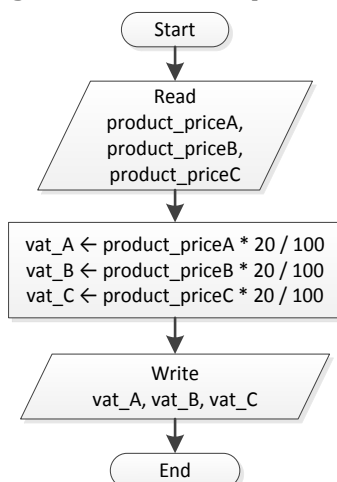


Figure 5-1 Calculating the 20% VAT for three products without the use of a constant

Even though this algorithm is absolutely correct, the problem is that the author used the 20% VAT (20/100) three times. If this were an actual computer program, the CPU would be forced to calculate the result of the division (20/100) three individual times.

Notice: Generally speaking, division and multiplication are time consuming operations that must be avoided when possible.

A much better solution would be to use a variable, as shown in **Figure 5-2**. This reduces the number of division and multiplication operations and also decreases the potential for typographical errors.

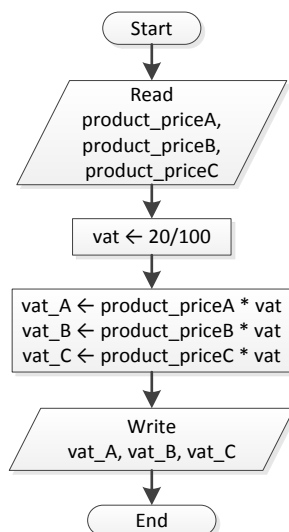


Figure 5-2 Calculating the 20% VAT for three products using a variable, `vat`

This time the division (20/100) is calculated only once, and then its result is used to calculate the VAT of each product.

But even now, the algorithm (which might later become a computer program) isn't perfect; `vat` is a variable and any programmer could accidentally change its value.

The ideal solution would be to change the variable `vat` to a constant `VAT`, as shown in **Figure 5-3**.

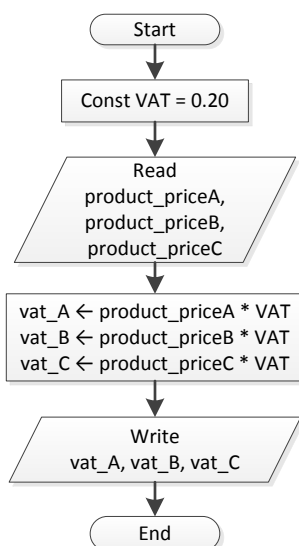


Figure 5-3 Calculating the 20% VAT for three products using a constant, `VAT`

Notice: Please note that when a constant is declared, the equals symbol (=) is used instead of the left arrow.

This last solution is the best choice for many reasons.

- No one, including the programmer, can change the value of `VAT` just by accidentally writing a statement such as `VAT ← 0.60` in any position of the program.
- The potential for typographical errors is minimized.
- The number of multiplication and division operations is kept as low as possible.
- If one day the finance minister decides to increase the Value Added Tax from 20% to 22%, the programmer needs to change just one line of code!

5.3 How Many Types of Variables and Constants Exist in PHP?

Many different types of variables and constants exist in all computer languages. The reason for this diversity is the different types of data each variable or constant can hold. Most of the time, variables and constants hold the following types of data.

- **Integers:** Integer values are positive or negative numbers without any fractional part, such as 5, 100, 135, -25, and -5123.
- **Reals:** Real values are positive or negative numbers that include a fractional part, such as 5.14, 7.23, 5.0, 3.14, and -23.78976. Real values are also known as “floats.”
- **Booleans**¹: Boolean variables or constants can hold only one of two values: `true` or `false`.
- **Characters:** Characters are alphanumeric values (always enclosed in single or double quotes) such as “Hello Zeus”, ‘I am 25 years old’, or ‘Peter Loves Jane For Ever’. Character values are also known as “strings”!!! (Probably the word “string” makes you visualize something wearable, but unfortunately it’s not. Please keep your dirty precious mind focused on computer science!)

5.4 Rules for Naming Variables in PHP

Certain rules must be followed when assigning names to your variables.

- PHP variables are written as `$var_name`, where `var_name` is the name of the variable. The name can contain only Latin characters (English uppercase or lowercase characters), numbers, and the underscore character (`_`). Examples of variable names are `$firstName`, `$last_name1`, and `$age`.
- Variable names are case sensitive, which means there is a distinct difference between uppercase and lowercase characters. For example, `$myVAR`, `$myvar`, `$MYVAR`, and `$MyVar` are actually four different variables.

¹ George Boole (1815–1864) was an English mathematician, philosopher, and logician. He is best known as the architect of what is now called Boolean logic (Boolean algebra), the basis of the modern digital computer.

- No space characters are allowed. If a variable is described by more than one word, you can use the underscore character (`_`) between the words. For example, the variable name `$student age` is wrong. Instead, you might use `$student_age`, or even `$studentAge`.
- A valid variable name can start with a letter or an underscore. Numbers are allowed, but they cannot be used at the beginning of the variable name. For example the variable name `$1student_name` is not properly written. Instead, you might use something like `$student_name1` or `$student1_name`.
- A variable name is usually chosen in a way that describes the meaning and the role of its contained data. For example, a variable that holds a temperature value might be named `$temperature`, `$temp`, or even `$t`.

5.5 Rules for Naming Constants in PHP

Certain rules must be followed when assigning names to your constants.

- PHP constants are written as `CONST_NAME`, where `CONST_NAME` is the name of the constant. The name can contain only Latin characters (English uppercase or lowercase characters), numbers, and the underscore character (`_`). Please note that PHP constants do not include the prefix `$` as variables do. Moreover, even though lowercase letters are permitted, it is advisable to use only uppercase letters. This helps you to visually distinguish constants from variables. Examples of constant names are `VAT` and `COMPUTER_NAME`.
- Most of the time, constant names are case sensitive, which means there is a distinct difference between uppercase and lowercase characters. For example, the `myCONST`, `myconst`, `MYCONST`, and `MyConst` are actually four different constants.
- No space characters are allowed. If a constant is described by more than one word, you can use the underscore character (`_`) between the words. For example, the constant name `COMPUTER NAME` is wrong. Instead, you might use `COMPUTER_NAME`, or even `COMPUTERNAME`.
- A valid constant name can start with a letter or an underscore. Numbers are allowed, but they cannot be used at the beginning of the constant name. For example, the constant name `1COMPUTER_NAME` is not properly written. Instead, you might use something like `COMPUTER_NAME1` or `COMPUTER1_NAME`.
- A constant name is usually chosen in a way that describes the meaning and the role of its contained data. For example, a constant that holds the Value Added Tax might be named `VAT`, or `VALUE_ADDED_TAX`.

5.6 What Does the Phrase “Declare a Variable” Mean?

Declaration is the process of reserving a portion in main memory (RAM) for storing the contents of a variable. In many high-level computer languages, the programmer must write a specific statement to reserve that portion in the RAM. In most cases, they even need to specify the variable type so that the compiler or the interpreter knows exactly how much space to reserve.

Here are some examples showing how variables are declared in different high-level computer languages.

Declaration Statement	High-level Computer Language
Dim sum as Integer	Visual Basic
int sum;	Visual C#, C++, Java, and many more
sum: Integer;	Pascal, Delphi
var sum;	Javascript

5.7 How to Declare Variables in PHP

In PHP, there is no need to declare variables as you would in C# or C++. Variables are declared when first used. For example, the statement

```
$number1 = 0;
```

declares the variable `$number1` and initializes it to 0.

Notice: In PHP, you can represent the action of assigning a value to a variable by using the equals (=) sign. This is equivalent to the left arrow in flowcharts.

5.8 How to Declare Constants in PHP

You can declare constants in PHP using the `define` keyword.

```
define("name", value);
```

The following examples show how to declare constants in PHP.

```
define("VAT", 0.22);
define("COMPUTER_NAME", "pc01");
define("FAVORITE_SONG", "We are the world");
```

Notice: Please note that PHP requires that all statements be terminated with a semicolon (;).

Once a constant is defined, its value cannot be changed while the script is running.

5.9 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A variable is a location in the computer's secondary storage device.
2. For a value assignment operator in a flowchart, you can use either a left or a right arrow.
3. A variable can change its content while the script executes.
4. A constant can change its content while the script executes.
5. The value 10.0 is an integer.
6. A Boolean variable can hold only one of two values.
7. The value "10.0" enclosed in double quotes is a real value.

8. In computer science, a string is something that you can wear.
9. The name of a variable can contain numbers.
10. A variable can change its name while the script executes.
11. The name of a variable cannot be a number.
12. The name of a constant must always be a descriptive one.
13. The name `$student name` is not a valid variable name.
14. The name `$student_name` is a valid constant name.
15. In PHP, the name of a constant can contain uppercase and lowercase letters.
16. In PHP, you must declare a variable.
17. In PHP, you must always declare at least one constant.

5.10 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. A variable is a place in
 - a. a hard disk.
 - b. a DVD disc.
 - c. a USB flash drive.
 - d. all of the above
 - e. none of the above
2. A variable can hold
 - a. one value at a time.
 - b. many values at a time.
 - c. all of the above
 - d. none of the above
3. Using constants in a PHP script
 - a. helps programmers to completely avoid typographical errors.
 - b. helps programmers to avoid using division and multiplication.
 - c. all of the above
 - d. none of the above
4. Which of the following is an integer?
 - a. 5.0
 - b. -5
 - c. "5"
 - d. none of the above is an integer.
5. A Boolean variable can hold the value
 - a. one.
 - b. zero.
 - c. true.

- d. "true".

e. none of the above
6. In PHP, strings are

a. enclosed in single quotes.

b. enclosed in double quotes.

c. both of the above
7. Which of the following is **not** a valid PHP variable?

a. \$city_name

b. \$cityName

c. \$cityname

d. \$city-name
8. You can define a constant by using the `define()` statement. Once a constant is defined,

a. it can never be changed.

b. it can be changed using the `define()` statement again.

c. Both of the above are correct.

5.11 Review Exercises

Complete the following exercises.

1. Match each element from the first column with one element from the second column.

Value	Data Type
1. "True"	a. Boolean
2. 123	b. Real
3. False	c. String
4. 10.0	d. Integer

2. Match each element from the first column with one element from the second column.

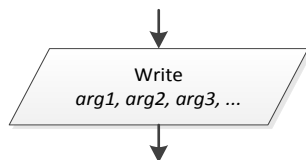
Data	Data Type
1. The name of a person	a. Boolean
2. The age of a person	b. Real
3. The result of the division 5/2	c. Integer
4. Is it black or is it white?	d. String

Chapter 6

Handling Input and Output

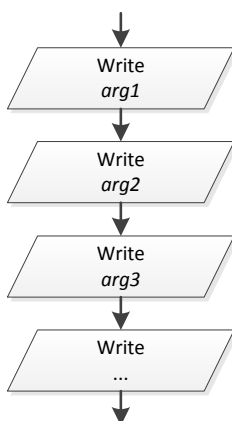
6.1 Which Statement Outputs Messages and Results on a User's Screen?

A flowchart uses the oblique parallelogram and the reserved word “Write” to display a message or the final results to the user's screen.



where *arg1*, *arg2*, and *arg3* can be variables, expressions, or even strings enclosed in double quotes.

The oblique parallelogram that you have just seen is equivalent to



In PHP, you can achieve the same result by using the statement

```
echo arg1, arg2, arg3, ... ;
```

or the equivalent sequence of statements.

```
echo arg1;
echo arg2;
echo arg3;
...
```

Remember! If you want to display a string on the screen, the string must be enclosed in double quotes.

The following two statements:

```
<?php
$a = 5 + 6;
echo "The sum of 5 and 6 is ", $a;
```

```
?>
```

display the message shown in **Figure 6-1**.

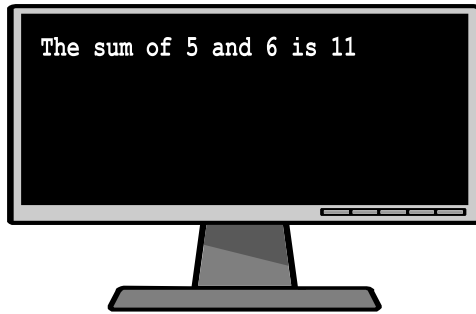


Figure 6-1 A string displayed on the screen

Notice: Please note the space inserted at the end of the first string in PHP, just after the word "is." If you remove it, the number 11 will get too close to the last word and the output on the screen will be

The sum of 5 and 6 is11

You can also calculate the result of a mathematical expression directly in an `echo` statement. The following statement:

```
<?php
    echo "The sum of 5 and 6 is ", 5 + 6;
?>
```

displays exactly the same message as the statements in **Figure 6-1**.

Notice: Please note that PHP requires that all statements be terminated with a semicolon.

6.2 How to Output Special Characters

Look carefully at the following example:

```
<?php
    echo "Good Morning";
    echo "Good Evening";
    echo "Good Night";
?>
```

Although you may believe that these three messages are displayed one under the other, the actual output result is shown in **Figure 6-2**.



Figure 6-2 The output result displays on one line

In order to output a “line break” you must put the special sequence of characters `\n` after every sentence.

```
<?php
echo "Good Morning\n";
echo "Good Evening\n";
echo "Good Night\n";
?>
```

The output result now appears in **Figure 6-3**.



Figure 6-3 The output result now displays line breaks

Keep in mind that the same result can be accomplished with one single statement.

```
echo "Good Morning\nGood Evening\nGood Night\n";
```

Another interesting sequence of characters is the `\t` which can be used to output a “tab stop.” The tab character (`\t`) is useful for aligning output.

```
<?php
echo "Good Morning\t";
echo "Good Evening\n";
echo "Good Afternoon\t";
echo "Good Night";
?>
```

The output result now appears in **Figure 6-4**.

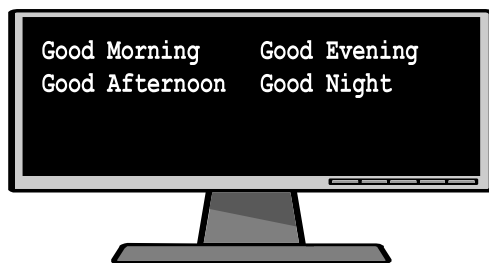


Figure 6-4 The output result now displays tabs

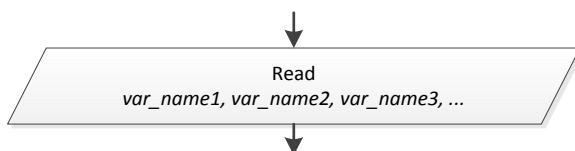
Of course, the same result can be accomplished with one single statement.

```
echo "Good Morning\tGood Evening\nGood Afternoon\tGood Night";
```

6.3 Which Statement Lets the User Enter Data?

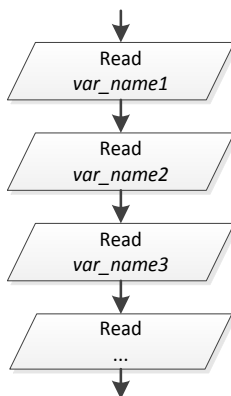
Do you recall the three main stages involved in creating an algorithm or a computer program? The first stage was the “data input” stage, in which the computer lets the user enter data such as numbers, their name, their address, or their year of birth.

A flowchart uses the oblique parallelogram and the reserved word “Read” to let a user enter his or her data.



where *var_name1*, *var_name2*, and *var_name3* must be variables only.

The oblique parallelogram that you have just seen is equivalent to



Notice: When a `Read` statement is executed, the flow of execution is interrupted until the user has entered all the data. When data entry is complete, the flow of execution continues to the next statement. Usually data are entered from a keyboard.

In PHP, data input is accomplished using the PHP statement

```
$var_name = trim(fgets(STDIN));
```

where

- `$var_name` can be any variable.
- `STDIN` stands for STANDARD INPUT (this is usually the keyboard).
- function `trim()` is discussed later in this book.

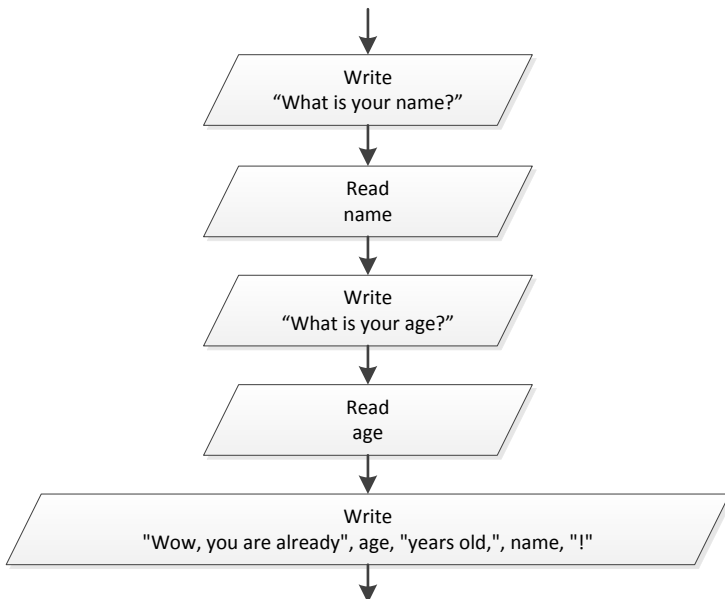
The following code fragment lets the user enter his or her name and age.

```
$name = trim(fgets(STDIN));  
$age = trim(fgets(STDIN));  
echo "Wow, you are already", age, "years old,", name, "!";
```

It could be even better if, before each data input, a “prompt” message is displayed. This makes the script more user-friendly. For example, look at the following script.

```
echo "What is your name?";  
$name = trim(fgets(STDIN));  
  
echo "What is your age?";  
$age = trim(fgets(STDIN));  
  
echo "Wow, you are already", age, "years old,", name, "!";
```

The corresponding flowchart fragment looks like this.



Now when a user executes your script he or she knows exactly what to enter.

Notice: In this book there is a subtle difference between the words “prompts” and “lets.” When the wording of an exercise says “Write a PHP script that **prompts** the user to enter...” this means that you must include a prompt message before every data input. On the other hand, when the wording of an exercise says “Write a PHP script that **lets** the user enter...” this means that you are not actually required to include a prompt message; that is, it is not wrong to include one but you don’t have to!

6.4 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. In PHP, the word `fgets` is a reserved word.
2. The `echo` statement can be used to display a message or the content of a variable.
3. When the `trim(fgets(STDIN))` statement is executed, the flow of execution is interrupted until the user has entered a value.
4. One single statement, `trim(fgets(STDIN))`, can be used to enter multiple data values.
5. Before data input, a prompt message must always be displayed.

6.5 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The statement `echo "Hello"` displays
 - a. the word `Hello` (without the double quotes).
 - b. the word `"Hello"` (including the double quotes).
 - c. the content of the variable `Hello`.
 - d. none of the above
2. The statement `echo "Hello\nHermes"` displays
 - a. the message `Hello Hermes`.
 - b. the word `Hello` in one line and the word `Hermes` in the next one.
 - c. the message `HelloHermes`.
 - d. the message `Hello\nHermes`.
 - e. none of the above
3. The statement `$data1_data2 = trim(fgets(STDIN))`
 - a. lets the user enter a value and assigns it to variable `$data1`. Variable `$data2` remains empty.
 - b. lets the user enter a value and assigns it to variable `$data1_data2`.
 - c. lets the user enter two values and assigns them to variables `$data1` and `$data2`.
 - d. none of the above

Chapter 7

Operators

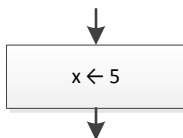
7.1 The Value Assignment Operator

The most commonly used operator in PHP is the value assignment operator (=). For example, the PHP statement

```
$x = 5;
```

assigns a value of 5 to variable \$x.

As you read in Chapter 5, this is equivalent to the left arrow used in flowcharts.



Probably the left arrow used in a flowchart is more convenient and clearer than the value assignment operator (=) sign because it shows in a more graphical way that the value or the result of an expression on the right is assigned to a variable on the left.

Be careful! The (=) sign is not equivalent to the one used in mathematics. In mathematics, the expression $x = 5$ is read as “*x is equal to 5.*” However, in PHP the expression `$x = 5` is read as “*assign the value 5 to x*” or “*set x equal to 5.*” They look the same but they act differently!

In mathematics, the following two lines are equivalent:

```
x = y + z
y + z = x
```

The first one can be read as “*x is equal to the sum of y and z*” and the second one as “*the sum of y and z is equal to x.*”

On the other hand, in PHP, the following two statements are definitely **not** equivalent. In fact, the second one is considered wrong!

```
$x = $y + $z;
$y + $z = $x;
```

The first statement seems quite correct. It can be read as “*Set x equal to the sum of y and z*” or “*Assign the sum of y and z to x.*”

But what about the second one? Think! Is it possible to assign the value of x to $y + z$? The answer is obviously a big “NO!”

Remember! In PHP, the variable on the left side of the (=) sign represents a region in main memory (RAM) where a value can be stored. Thus, on the left side only one single variable must exist! However, on the right side there can be a number, a variable, a string, or even a complex mathematical expression.

In **Table 7-1** you can find some examples of value assignments.

Table 7-1 Examples of Value Assignments

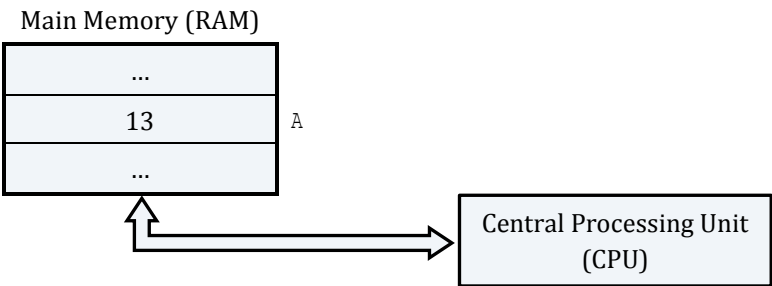
<code>\$a = 9</code>	Assign a value of 9 to variable <code>\$a</code> .
<code>\$b = \$c</code>	Assign the content of variable <code>\$c</code> to variable <code>\$b</code> .
<code>\$d = "Hello Zeus"</code>	Assign the string <i>Hello Zeus</i> to variable <code>\$d</code> .
<code>\$d = \$a + \$b</code>	Calculate the sum of the contents of variables <code>\$a</code> and <code>\$b</code> and assign the result to variable <code>\$d</code> .
<code>\$b = \$a + 1</code>	Calculate the sum of the content of variable <code>\$a</code> and 1 and assign the result to variable <code>\$b</code> . Please note that the content of variable <code>\$a</code> is not altered.
<code>\$a = \$a + 1</code>	Calculate the sum of the content of variable <code>\$a</code> and 1 and assign the result back to variable <code>\$a</code> . In other words, increase variable <code>\$a</code> by one.

Confused about the last one? Are you thinking about your math teacher right now? What would he/she say if you had written `a = a + 1` on the blackboard? Can you personally think of a number that is equal to the number itself plus one? Are you nuts? This means that 5 is equal to 6 and 10 is equal to 11!

Obviously, things are different in computer science. The statement `$a = $a + 1` is absolutely acceptable! It instructs the CPU to retrieve the value of variable `$a` from main memory (RAM), to increase the value by one, and to assign the result back to variable `$a`. The old value of variable `$a` is replaced by the new one.

Still don't get it? Let's take a look at how the CPU and main memory (RAM) cooperate with each other in order to execute the statement $A \leftarrow A + 1$ (this is the same as the statement `$a = $a + 1` in PHP).

Let's say that there is a region in memory, named `A` and it contains the number 13



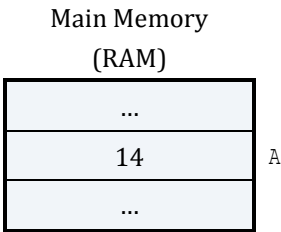
When a program instructs the CPU to execute the statement

$$A \leftarrow A + 1$$

the following procedure is executed:

- the number 13 is transferred from the RAM's region named `A` to the CPU;
- the CPU calculates the sum of 13 and 1; and
- the result, 14, is transferred from the CPU to the RAM's region named `A` replacing the existing number, 13.

After execution, the RAM looks like this.



7.2 Arithmetic Operators

Just like every high-level programming language, PHP supports almost all types of arithmetic operators.

Arithmetic Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder after integer division)

The first four operators are straightforward and need no further explanation. The modulus operator (%) returns the remainder of an integer division, which means that

```
$a = 13 % 3
```

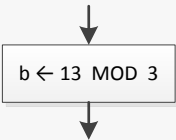
assigns a value of 1 to variable \$a.

The modulus operator (%) can be used with floating-point numbers as well, but the result is always an integer. For example, the operation

```
$b = 13.4 % 3
```

assigns a value of 1 (and not 1.4, as you may mistakenly expect) to variable \$b.

Notice: Please keep in mind that flowcharts are a loose method used to represent an algorithm. Although the use of the modulus (%) operator is allowed in flowcharts, this book uses the commonly accepted MOD operator instead! For example, the PHP statement `$b = 13 % 3` is represented in a flowchart as



In mathematics, as you already know, you are allowed to use parentheses as well as braces and brackets.

$$y = \frac{5}{2} \left\{ 3 + 2 \left[4 + 7 \left(7 - \frac{4}{3} \right) \right] \right\}$$

However, in PHP there is no such thing as braces and brackets. Parentheses are all you have; therefore, the same expression should be written using parentheses instead of braces or brackets.

```
$y = 5 / 2 * (3 + 2 * (4 + 7 * (7 - 4 / 3)));
```

One other thing that is legal in mathematics but not in PHP is that you can skip the multiplication operator and write $3x$, meaning “3 times x .” In PHP, however, you must always use an asterisk anywhere a multiplication operation exists. This is one of the most common mistakes novice programmers make when they write mathematical expressions in PHP.

7.3 What is the Precedence of Arithmetic Operators?

Arithmetic operators follow the same precedence rules as in mathematics, and these are: multiplication and division are performed first, addition and subtraction are performed afterwards.

Higher Precedence  Lower precedence	Arithmetic Operators
	$*, /, \%$
	$+, -$

When multiplication and division exist in the same expression, and since both are of the same precedence, they are performed left to right (the same way as you read), which means that the expression

```
$y = 6 / 3 * 2
```

is equivalent to $y = \frac{6}{3} \cdot 2$, and assigns a value of 4 to variable y , (division is performed before multiplication).

If you want, however, the multiplication to be performed before the division, you can use parentheses to change the precedence, that is:

```
$y = 6 / (3 * 2)
```

which is equivalent to $y = \frac{6}{3 \cdot 2}$

Notice: PHP scripts can be written using your computer’s text editor application, but keep in mind that it is not possible to write fractions in the form of $\frac{6}{3}$ or $\frac{4x+5}{6}$. Forget it! There is no equation editor in a text editor! All fractions must be written on one single line. For example, $\frac{6}{3}$ should be written as $6/3$, and $\frac{4x+5}{6}$ should be written as $(4*x+5)/6$.

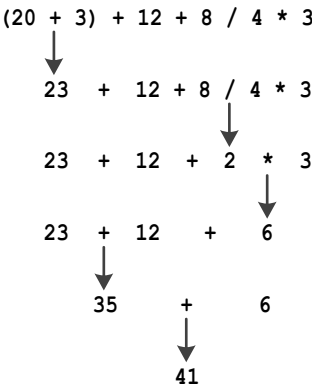
- The order of operations can be summarized as follows:
1. Any operations enclosed in parentheses are performed first.
 2. Next, any multiplication and division operations are performed from left to right.
 3. In the end, any addition and subtraction operations are performed from left to right.

So, in statement `$y = (20 + 3) + 12 + 8 / 4 * 3;`

the operations are performed as follows:

1. 20 is added to 3, yielding a result of 23.
2. 8 is divided by 4, yielding a result of 2. This result is then multiplied by 3, yielding a result of 6.
3. 23 is added to 12, yielding a result of 35. This result is then added to 6, yielding a final result of 41.

Next is the same sequence of operations presented in a more graphical way.



7.4 Compound Assignment Operators

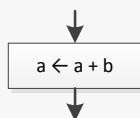
PHP offers a special set of operators known as compound assignment operators, which can help you write code faster.

Operator	Description	Example	Equivalent to
<code>+=</code>	Addition assignment	<code>\$a += \$b</code>	<code>\$a = \$a + \$b</code>
<code>-=</code>	Subtraction assignment	<code>\$a -= \$b</code>	<code>\$a = \$a - \$b</code>
<code>*=</code>	Multiplication assignment	<code>\$a *= \$b</code>	<code>\$a = \$a * \$b</code>
<code>/=</code>	Division assignment	<code>\$a /= \$b</code>	<code>\$a = \$a / \$b</code>
<code>%=</code>	Modulus assignment	<code>\$a %= \$b</code>	<code>\$a = \$a % \$b</code>

Looking at the “Equivalent to” column, it becomes clear that same result can be achieved by just using the classic assignment (`=`) operator. So the question that arises here is *why do these operators exist?*

The answer is simple: It's a matter of convenience. Once you start using them, your life finds a different meaning!

Notice: Please keep in mind that flowcharts are a loose method to represent an algorithm. Although the use of compound assignment operators is allowed in flowcharts, this book uses only the commonly accepted operators shown in the "Equivalent to" column. For example, the PHP statement `$a += $b` is represented in a flowchart as



Exercise 7.4-1 Which PHP Statements are Syntactically Correct?

Which of the following PHP assignment statements are syntactically correct?

i. <code>\$a = -10;</code>	v. <code>\$a = COWS;</code>	ix. <code>\$a = true;</code>
ii. <code>10 = \$b;</code>	vi. <code>\$a + \$b = 40;</code>	x. <code>\$a %= 2;</code>
iii. <code>\$a_b = \$a_b + 1;</code>	vii. <code>\$a = 3 \$b;</code>	xi. <code>\$a += 1;</code>
iv. <code>\$a = "COWS";</code>	viii. <code>\$a = "true";</code>	xii. <code>\$a =* 2;</code>

Solution

- i. **Correct.** It assigns the integer value -10 to variable `$a`.
- ii. **Wrong.** On the left side of the assignment operator, only variables can exist.
- iii. **Correct.** It increases variable `$a_b` by one.
- iv. **Correct.** It assigns the string (the text) `COWS` to variable `$a`.
- v. **Correct.** It assigns the content of constant `COWS` to variable `$a`.
- vi. **Wrong.** On the left side of the assignment operator, only one variable can exist.
- vii. **Wrong.** It should have been written as `$a = 3 * $b`.
- viii. **Correct.** It assigns the string `true` to variable `$a`.
- ix. **Correct.** It assigns the value `true` to variable `$a`.
- x. **Correct.** This is equivalent to `$a = $a % 2`.
- xi. **Correct.** This is equivalent to `$a = $a + 1`.
- xii. **Wrong.** It should have been written as `$a *= 2` (which is equivalent to `$a = $a * 2`).

Exercise 7.4-2 Finding Variable Types

What is the type of each of the following variables?

- i. \$a = 15;
- iii. \$b = "15";
- v. \$b = true;
- ii. \$width = "10 meters";
- iv. \$temp = 13.5;
- vi. \$b = "true";

Solution

- i. The value 15 belongs to the set of integers, thus the variable \$a is an integer.
- ii. The “10 meters” is text, thus the \$width variable is a string.
- iii. The “15” is text string, thus the \$b variable is a string.
- iv. The value 13.5 belongs to the set of real numbers, thus the variable \$temp is real.
- v. The value true is Boolean, thus variable \$b is a Boolean.
- vi. The value true is a text, thus variable \$b is a string.

7.5 Incrementing/Decrementing Operators

Adding 1 to a number, or subtracting 1 from a number, are both so common in computer programs that PHP incorporates a special set of operators to do this. PHP supports two types of incrementing and decrementing operators:

- pre-incrementing/decrementing operators
- post-incrementing/decrementing operators

Pre-incrementing/decrementing operators are placed before the variable name. Post-incrementing/decrementing operators are placed after the variable name. These four types of operators are shown here.

Operator	Description	Example	Equivalent to
Pre-incrementing	Increment a variable by one	++\$a;	\$a = \$a + 1
Pre-decrementing	Decrement a variable by one	--\$a;	\$a = \$a - 1
Post-incrementing	Increment a variable by one	\$a++;	\$a = \$a + 1
Post-decrementing	Decrement a variable by one	\$a--;	\$a = \$a - 1

Once again, looking at the “Equivalent to” column, it becomes clear that you can achieve the same result by just using the classic assignment operator (=). However, it is much more sophisticated and productive to use these new operators. In the beginning you may hate them, but as the years pass you are definitely gonna love them!

Let’s see an example with a pre-incrementing operator,

```
<?php
$a = 5;
++$a;           //This is equivalent to $a = $a + 1
```

```
$b = $a;
echo $a, ", ", $b;
?>
```

and another one with a post-incrementing operator.

```
<?php
$a = 5;
$a++;           //This is equivalent to $a = $a + 1
$b = $a;
echo $a, ", ", $b;
?>
```

In both examples a value of 6 is assigned to variable `$b`! So, where is the catch? Are these two examples equivalent? The answer is “yes,” but only in these two examples. In other cases the answer will likely be “no.” There is a small difference between them.

Let’s spot that difference! The rule is that a pre-increment/decrement operator performs the increment/decrement operation first and then delivers the new value. A post-increment/decrement operator delivers the old value first and then performs the increment/decrement operation. Look carefully at the next two examples.

```
$a = 5;
$b = ++$a;
echo $a, ", ", $b;           //Outputs: 6, 6
```

and

```
$a = 5;
$b = $a++;
echo $a, ", ", $b;           //Outputs: 6, 5
```

In the first example, variable `$a` is incremented by one and then its new value is assigned to variable `$b`. In the end, both variables contain a value of 6.

In the second example, the value 5 of variable `$a` is assigned to variable `$b`, and then variable `$a` is incremented by one. In the end, variable `$a` contains a value of 6 but variable `$b` contains a value of 5!

Notice: The double slashes after the `echo` statements indicate that the text that follows is a comment; thus, it is never executed.

7.6 String Operators

There are two operators that can be used to concatenate (join) strings.

Operator	Description	Example	Equivalent to
.	Concatenation	<code>\$a = "Hi" . " there";</code>	
<code>.=</code>	Concatenation assignment	<code>\$a .= "Hello";</code>	<code>\$a = \$a . "Hello";</code>

Notice: *Joining two separate strings into a single one is called “concatenation.”*

The following example displays “What’s up, dude?”

```
<?php
    $a = "What's ";
    $b = "up, ";
    $c = $a . $b;
    $c .= "dude?";

    echo $c;
?>
```

Exercise 7.6-1 *Concatenating Names*

Write a PHP script that prompts the user to enter his or her first and last name (assigned to two different variables) and then joins them together in a single string (concatenation).

Solution

The PHP script is shown here.

```
<?php
    echo "Enter first name: ";
    $first_name = trim(fgets(STDIN));
    echo "Enter last name: ";
    $last_name = trim(fgets(STDIN));

    $full_name = $first_name . " " . $last_name;
    echo $full_name;
?>
```

Notice: *Please note the extra space character added between the first and last name*

7.7 **Review Questions: True/False**

Choose **true** or **false** for each of the following statements.

1. The statement `$x = 5` can be read as “*Variable \$x is equal to 5.*”
2. The value assignment operator assigns the result of an expression to a variable.
3. A value can be assigned to a variable only by using the `trim(fgets(STDIN))` statement.
4. The statement `5 = $y` assigns value 5 to variable `$y`.

5. On the right side of a value assignment operator an arithmetic operator must always exist.
6. On the left side of a value assignment operator more than one variable can exist.
7. You cannot use the same variable on both sides of a value assignment operator.
8. The statement `$a = $a + 1` decrements variable `$a` by one.
9. In PHP, the word `MOD` is a reserved word.
10. The statement `$x = 0 % 5` assigns a value of 5 to variable `$x`.
11. The operation `5 % 0` is not possible.
12. Division and multiplication have the higher precedence among the arithmetic operators.
13. When division and multiplication operators co-exist in an expression, multiplication operations are performed before division.
14. The expression `8 / 4 * 2` is equal to 1.
15. The expression `4 + 6 / 6 + 4` is equal to 9.
16. The expression `$a + $b + $c / 3` calculates the average value of three numbers.
17. The statement `$a += 1` is equivalent to `$a++`.
18. The statement `$a = "true"` assigns a Boolean value to variable `$a`.
19. The statement `$a = 2 * $a` doubles the content of variable `$a`.
20. The statements `$a++` and `++$a` are completely equivalent.
21. The statement `$a = $a++ - $a--`; always assigns a value of -1 to variable `$a`.
22. The statement `$a = "George" . "Malkovich"` assigns value `GeorgeMalkovich` to variable `$a`.
23. The following code fragment satisfies the property of definiteness.

```
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
  
$x = $a / ($b - 7);  
echo $x;
```

7.8 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following PHP statements assigns a value of 10 to variable `$a`?
 - a. `10 = $a;`
 - b. `$a ← 10;`
 - c. `$a = 100/10;`
 - d. none of the above

2. The statement `$a = $b` can be read as
 - a. assign the content of variable `$a` to variable `$b`.
 - b. variable `$b` is equal to variable `$a`.
 - c. assign the content of variable `$b` to variable `$a`.
 - d. none of the above
3. The expression `0 % 10 + 2` is equal to
 - a. 7.
 - b. 2.
 - c. 12.
 - d. none of the above
4. Which of the following PHP statements is syntactically correct?
 - a. `$a = 4 * 2 $y - 8 / (4 * $q);`
 - b. `$a = 4 * 2 * $y - 8 / 4 * $q;`
 - c. `$a = 4 * 2 * $y - 8 / (4 */ $q);`
 - d. none of the above
5. Which of the following PHP statements increments variable `$a` by one and then assigns its value to variable `$b`?
 - a. `$b = $a + 1;`
 - b. `$b = $a++;`
 - c. `$b = ++$a;`
 - d. all of the above
6. Which of the following PHP statements assigns value George Malkovich to variable `$a`?
 - a. `$a = "George" . " " . "Malkovich";`
 - b. `$a = "George" . " Malkovich";`
 - c. `$a = "George " . "Malkovich";`
 - d. all of the above
7. The following code fragment

```
$x = 2;
$x++;
```

does **not** satisfy the property of

- a. finiteness.
- b. definiteness.
- c. effectiveness.
- d. none of the above

8. The following code fragment

```
$a = trim(fgets(STDIN));  
$x = 1 / $a;
```

does **not** satisfy the property of

- a. finiteness.
- b. input.
- c. effectiveness.
- d. none of the above

7.9 Review Exercises

Complete the following exercises.

1. Which of the following PHP assignment statements are syntactically correct?

- i. $\$a \leftarrow \$a + 1;$
 - ii. $\$a += \$b;$
 - iii. $\$a\ b = \$a\ b + 1;$
 - iv. $\$a = \$a++;$
 - v. $\$a = \text{hello};$
- vi. $\$a = 40";$
 - vii. $\$a = \$b \cdot 5;$
 - viii. $\$a = . "true";$
 - ix. $\$fdadstwsdgfgw = 1;$
 - x. $**\$a;$

2. What is the type of each of the following variables?

- i. $\$a = "false";$
 - ii. $\$w = \text{false};$
- iii. $\$b = "15\ \text{meters}";$
 - iv. $\$weight = "40";$
- v. $\$b = 13.0;$
 - vi. $\$b = 13;$

3. Match each element from the first column with one element from the second column.

Operation	Result
i. $1 / 2$	a. 100
ii. $1 / 2 * 2$	b. 0.25
iii. $0 \% 10 * 10$	c. 0
iv. $10 \% 2 + 7$	d. 0.5
	e. 7
	f. 1

4. What displays on the screen after executing each of the following code fragments?

i.

```
$a = 5;  
$b = $a * $a + 1;  
echo $b++;
```

ii.

```
$a = 9;  
$b = $a / 3 * $a;  
echo ++$b;
```

5. What displays on the screen after executing each of the following code fragments?

i.

```
$a = 5;
$a = $a++;
echo $a;
```

ii.

```
$a = 5;
$a = ++$a;
echo $a;
```

6. What is the result of each of the following operations?

i. $21 \% 5$

iv. $10 \% 6 \% 3$

ii. $10 \% 2$

v. $0 \% 3$

iii. $11 \% 2$

vi. $100 / 10 \% 3$

7. What displays on screen after executing each of the following code fragments?

i.

```
$a = 5;
$b = 2;
$c = $a \% ++$b;
$d = $b \% ($a + $b);
echo $c, " ", $d;
```

ii.

```
$a = 0.4;
$b = 8;
$a += 0.1;
$c = $a * $b \% $b;
echo $c;
```

8. Calculate the result of the expression $\$a \% \$b$ for the following cases.

i. $\$a = 20, \$b = 3$

iv. $\$a = 0, \$b = 3$

ii. $\$a = 15, \$b = 3$

v. $\$a = 3, \$b = 0$

iii. $\$a = 22, \$b = 3$

vi. $\$a = 2, \$b = 2$

9. Calculate the result of the expression

$$\$b * (\$a \% \$b) + \$a / \$b$$

for each of the following cases.

i. $\$a = 10, \$b = 5$

ii. $\$a = 10, \$b = 4$

10. What displays on the screen after executing the following code fragment?

```
$a = "My name is";
$a .= " ";
$a = $a . "George Malkovich";
echo $a;
```

11. Fill in the gaps in each of the following code fragments so that they display a value of 5.

i.

```
$a = 2;
$a = $a - ..... ;
echo $a;
```

ii.

```
$a = 4;
$b = $a * 0.5;
$b += $a;
$a = $b - ..... ;
echo $a;
```

12. What displays on the screen after executing the following code fragment?

```
$city = "California";  
$California = $city;  
echo $city, ", ", $California;
```

Chapter 8

Trace Tables

8.1 What is a Trace Table?

A trace table is a technique used to test algorithms or computer programs for logic errors that occur while the algorithm or program executes.

The trace table simulates the flow of execution. Statements are executed step by step, and the values of variables change as an assignment statement is executed.

Trace tables are typically used by novice programmers to help them visualize how a particular algorithm or program works. Trace tables can also help advanced programmers detect logic errors.

A typical trace table is shown here.

Step	Statement	Notes	\$variable1	\$variable2	\$variable3
1					
2					
...					

Let's see a trace table in action! For the following PHP script, a trace table is created to determine the values of the variables in each step.

```
<?php
    $x = 10;
    $y = 15;
    $z = $x * $y;
    $z++;
    echo $z;
?>
```

The trace table for this script is shown below. Notes are optional, but they help the reader to better understand what is really happening.

Step	Statement	Notes	\$x	\$y	\$z
1	\$x = 10	The value 10 is assigned to variable \$x.	10	?	?
2	\$y = 15	The value 15 is assigned to variable \$y.	10	15	?
3	\$z = \$x * \$y	The result of the product \$x * \$y is assigned to \$z.	10	15	150
4	\$z++	Variable \$z is incremented by one.	10	15	151
5	echo \$z	The value 151 is displayed.			

Exercise 8.1-1

Creating a Trace Table

What result is displayed when the following script is executed?

```
<?php
    $Ugly = "Beautiful";
    $Beautiful = "Ugly";
    $Handsome = $Ugly;
    echo "Beautiful";
    echo $Ugly;
    echo $Handsome;
?>
```

Solution

Let’s create a trace table to find the output result.

Step	Statement	Notes	\$Ugly	\$Beautiful	\$Handsome
1	\$Ugly = "Beautiful"	The string “Beautiful” is assigned to the variable \$Ugly.	Beautiful	?	?
2	\$Beautiful = "Ugly"	The string “Ugly” is assigned to the variable \$Beautiful.	Beautiful	Ugly	?
3	\$Handsome = \$Ugly	The value of variable \$Ugly is assigned to the variable \$Handsome.	Beautiful	Ugly	Beautiful
4	echo "Beautiful"	The string “Beautiful” is displayed.			
5	echo \$Ugly	The string “Beautiful” is displayed.			
6	echo \$Handsome	The string “Beautiful” is displayed.			

Exercise 8.1-2

Swapping Values of Variables

Write a PHP script that lets the user enter two values, in variables *\$a* and *\$b*. At the end of the program, the two variables should swap their values. For example, if variables *\$a* and *\$b* contain the values 5 and 7 respectively, after swapping their values, variable *\$a* should contain the value 7 and variable *\$b* should contain the value 5!

Solution

The following script, even though it may seem correct, is erroneous and doesn't really swap the values of variables \$a and \$b!

```
<?php
    $a = trim(fgets(STDIN));
    $b = trim(fgets(STDIN));

    $a = $b;
    $b = $a;

    echo $a, $b;

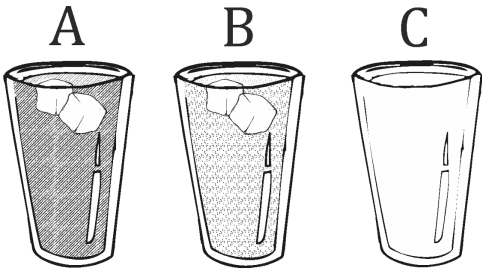
?>
```

Let's see why! Suppose the user enters two values, 5 and 7. The trace table is shown here.

Step	Statement	Notes	\$a	\$b
1	\$a = trim(fgets(STDIN))	User enters the value 5	5	?
2	\$b = trim(fgets(STDIN))	User enters the value 7	5	7
3	\$a = \$b	The value of variable \$b is assigned to variable \$a. Value 5 is lost!	7	7
4	\$b = \$a	The value of variable \$a is assigned to variable \$b	7	7
5	echo \$a, \$b	The values 7, 7 are displayed		

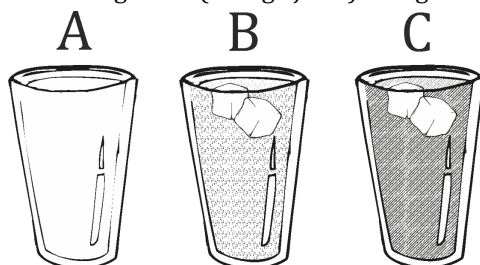
Oops! Where is the value 5?

The solution wasn't so obvious after all! So, how do you really swap values anyway? Consider two glasses: a glass of orange juice (called glass A), and a glass of lemon juice (called glass B). If you want to swap their content, all you must do is find and use one extra empty glass (called glass C).

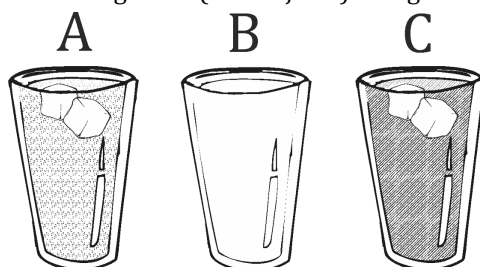


The steps that must be followed are:

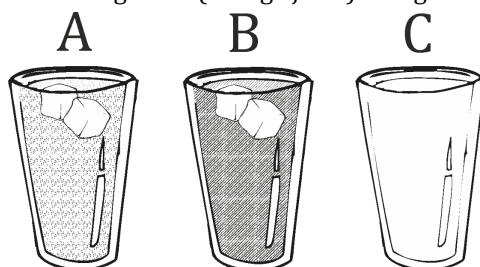
1. Empty the contents of glass A (orange juice) into glass C.



2. Empty the contents of glass B (lemon juice) into glass A.



3. Empty the contents of glass C (orange juice) into glass B.



Swapping completed successfully!

You can follow the same steps to swap the contents of two variables in PHP.

```
<?php
```

```
$a = trim(fgets(STDIN));
```

```
$b = trim(fgets(STDIN));
```

```
$c = $a; //Empty the contents of glass A (orange juice) into glass C
```

```
$a = $b; //Empty the contents of glass B (lemon juice) into glass A
```

```
$b = $c; //Empty the contents of glass C (orange juice) into glass B
```

```
echo $a, $b;
```

```
?>
```

Notice: Please note the double slashes after each of the three assignment statements. The text after double slashes is considered a comment and is never executed.

Exercise 8.1-3 *Swapping Values of Variables – A Second Approach*

Write a PHP script that lets the user enter two numeric values, in variables `$a` and `$b`. In the end, the two variables should swap their values.

Solution

Since the variables contain only numeric values, you can use the following PHP script.

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$a = $a + $b;
$b = $a - $b;
$a = $a - $b;

echo $a, ", ", $b;

?>
```

Notice: Compared to previous exercise, the only major disadvantage of this method is that it cannot swap the contents of alphanumeric variables (strings).

Exercise 8.1-4 *Creating a Trace Table*

Create a trace table to determine the values of the variables in each step of the PHP script for three different executions.

The input values for the three executions are: (i) 0.3, (ii) 4.5, and (iii) 10.

```
<?php
$b = trim(fgets(STDIN));
$c = 3;
$c = $c * $b;
$a = 10 * $c % 10;

echo $a;

?>
```

Solution

i. For the input value of 0.3, the trace table looks like this.

Step	Statement	Notes	\$a	\$b	\$c
1	<code>\$b = trim(fgets(STDIN))</code>	User enters value 0.3	?	0.3	?
2	<code>\$c = 3</code>		?	0.3	3
3	<code>\$c = \$c * \$b</code>		?	0.3	0.9
4	<code>\$a = 10 * \$c % 10</code>		9	0.3	0.9
5	<code>echo \$a</code>	The value 9 is displayed			

ii. For the input value of 4.5, the trace table looks like this.

Step	Statement	Notes	\$a	\$b	\$c
1	<code>\$b = trim(fgets(STDIN))</code>	User enters value 4.5	?	4.5	?
2	<code>\$c = 3</code>		?	4.5	3
3	<code>\$c = \$c * \$b</code>		?	4.5	13.5
4	<code>\$a = 10 * \$c % 10</code>		5	4.5	13.5
5	<code>echo \$a</code>	The value 5 is displayed			

iii. For the input value of 10, the trace table looks like this.

Step	Statement	Notes	\$a	\$b	\$c
1	<code>\$b = trim(fgets(STDIN))</code>	User enters value 10	?	10	?
2	<code>\$c = 3</code>		?	10	3
3	<code>\$c = \$c * \$b</code>		?	10	30
4	<code>\$a = 10 * \$c % 10</code>		0	10	30
5	<code>echo \$a</code>	The value 0 is displayed			

Exercise 8.1-5 *Creating a Trace Table*

Create a trace table to determine the values of the variables in each step when a value of 3 is entered.

```
<?php
    $a = trim(fgets(STDIN));

    $b = $a + 10;
    $a = $b * ($a - 3);
    $c = 3 * $b / 6;
    $d = $c * $c;
    $d--;
```

```
echo $d;
?>
```

Solution

For the input value of 3, the trace table looks like this.

Step	Statement	Notes	\$a	\$b	\$c	\$d
1	\$a = trim(fgets(STDIN))	User enters value 3	3	?	?	?
2	\$b = a + 10		3	13	?	?
3	\$a = \$b * (\$a - 3)		0	13	?	?
4	\$c = 3 * \$b / 6		0	13	6.5	?
5	\$d = \$c * \$c		0	13	6.5	42.25
6	\$d--		0	13	6.5	41.25
7	echo \$d	The value 41.25 is displayed				

8.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A trace table is a technique for testing a computer.
2. Trace tables help a programmer find errors in a computer program.
3. You cannot write a computer program without first creating its corresponding trace table.
4. In order to swap the values of two integer variables, you always need an extra variable.

8.3 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the PHP script for three different executions.

The input values for the three executions are: (i) 3, (ii) 4, and (iii) 1.

```
<?php
    $a = trim(fgets(STDIN));

    $a = ($a + 1) * ($a + 1) + 6 / 3 * 2 + 20;
    $b = $a % 13;
    $c = $b % 7;
    $d = $a * $b * $c;
    echo $a, " ", $b, " ", $c, " ", $d;

?>
```

2. Create a trace table to determine the values of the variables in each step of the PHP script for two different executions.

The input values for the two executions are: (i) 3, 4; and (ii) 4, 4

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$c = $a + $b;
$d = 1 + $a / $b * $c + 2;
$e = $c + $d;
$c += $d + $e;
$e--;
$d -= $c + $d % $c;
echo $c, ", ", $d, ", ", $e;
?>
```

Chapter 9

Using NetBeans IDE

9.1 Creating a New PHP Project

So far, you have learned some really good basics about PHP scripts. Now it's time to learn how to enter scripts into the computer, execute them, see how they perform, and see how they display the results.

The first thing you must do is create a new PHP project. The NetBeans IDE provides a wizard to help you do that. Start the NetBeans IDE, and from its main menu select “File→New Project...” as shown in **Figure 9-1**.

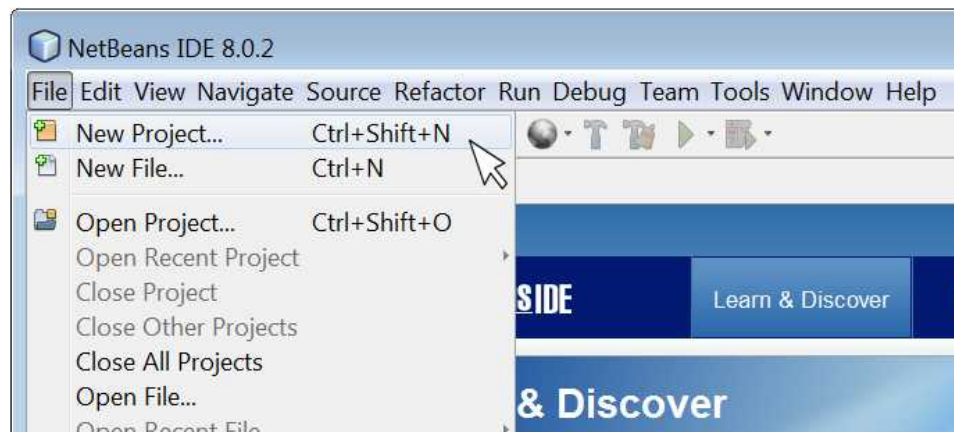


Figure 9-1 Starting a new PHP project from NetBeans IDE

Notice: A PHP project can contain many PHP scripts.

The “New Project” wizard will appear, and it will guide you through the steps that follow.

1. On the “New Project” screen of the wizard, under “Choose Project,” select “PHP Application” as shown in **Figure 9-2**. Then, click on the “Next” button.

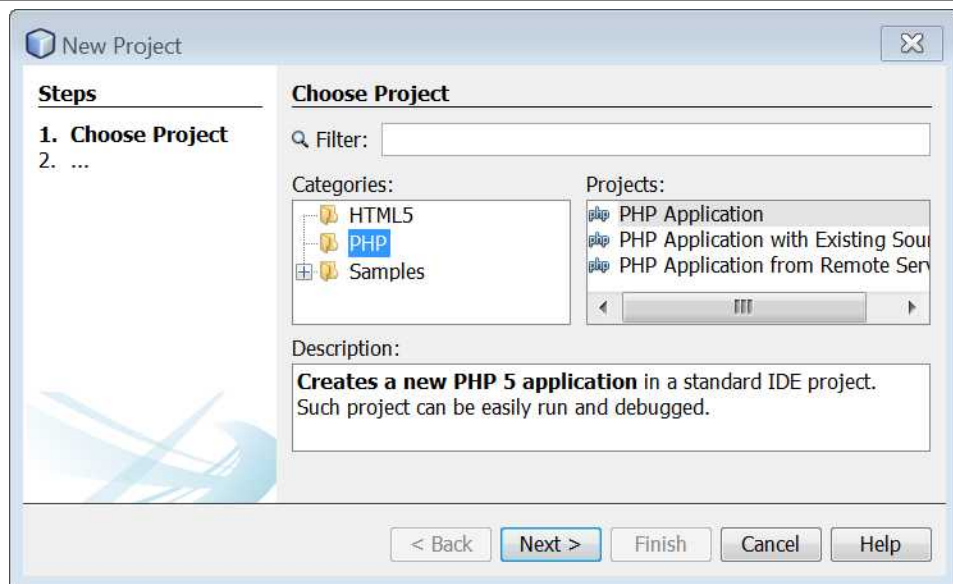


Figure 9-2 Selecting the PHP Application under Projects

2. You will see the “New PHP Project” screen of the wizard. Under “Name and Location” , in the “Project Name” field, type “testingProject” as shown in **Figure 9-3**. Leave all other fields unchanged. Then, click on the “Next” button.

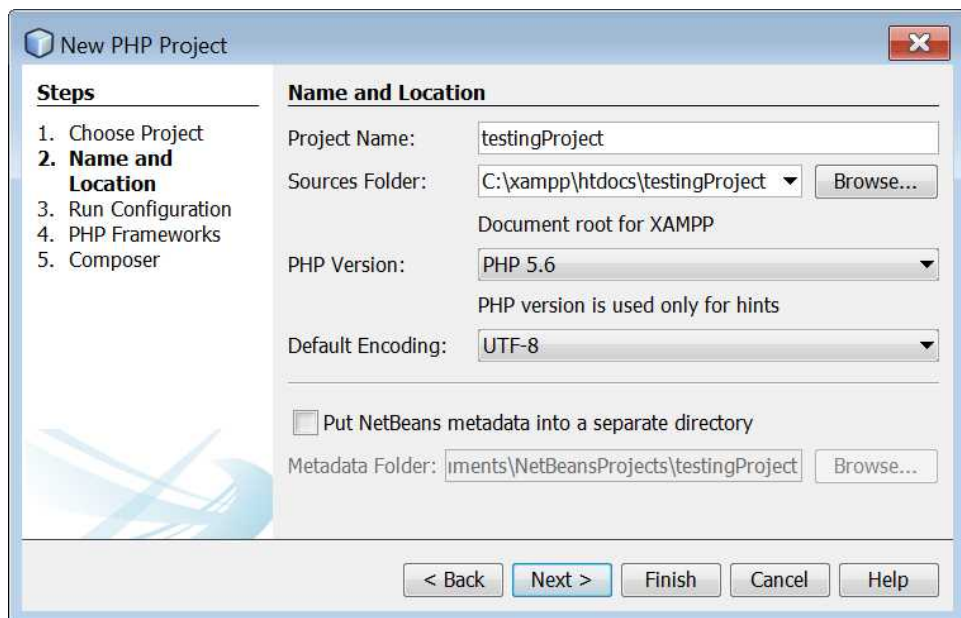


Figure 9-3 Selecting the name and location of a new PHP project

3. Under “Run Configuration”, from the “Run As” dropdown list, select “Script (run in command line)” as shown in **Figure 9-4**.

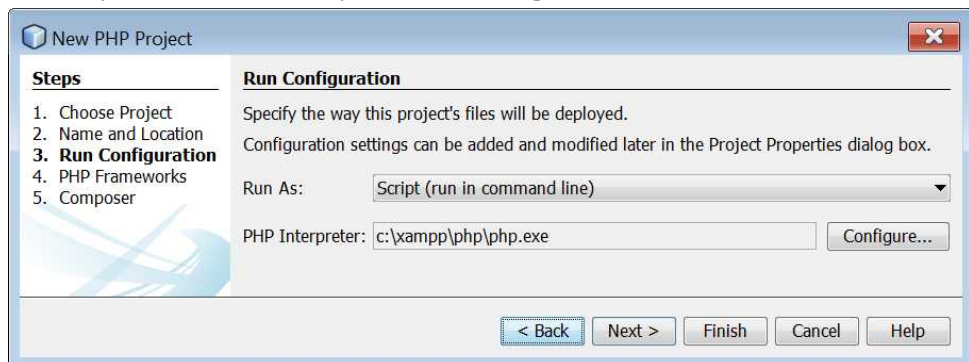


Figure 9-4 Selecting the configuration settings for a new PHP project

4. Click on the “Finish” button.

The project is created and opened in your NetBeans IDE. You should see the following components (see **Figure 9-5**):

- the Projects window, which contains a tree view of the components of the projects, including source files, libraries that your code may depend on, and so on
- the Source Editor window with a file called “index.php” open. In this file you can write your PHP code. Of course, one single project can contain many such files.

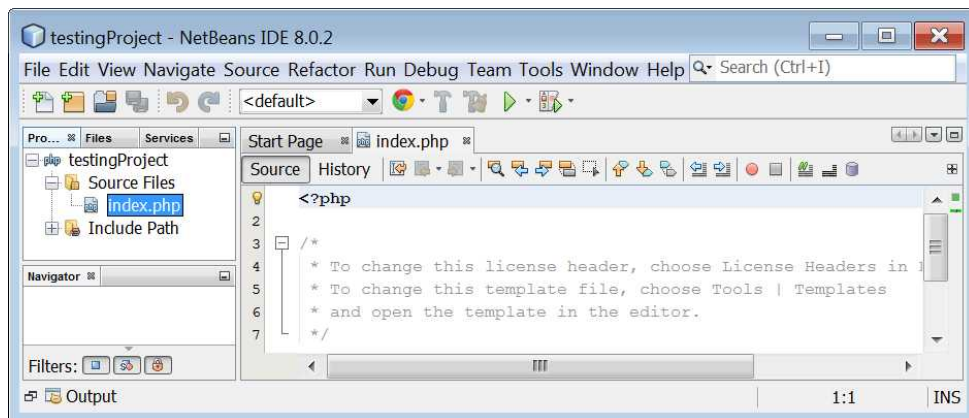


Figure 9-5 Viewing the “Projects” and “Source Editor” windows in NetBeans IDE

9.2 Writing and Executing a PHP Script

You have just seen how to create a new PHP project. Let’s now write the following (terrifying, and quite horrifying!) PHP script and try to execute it.

```
<?php
echo "Hello World";
```

?>

In window “index.php”, delete the commented lines (everything except `<?php`) and type only the first letter from the `echo` statement by hitting the “e” key on your keyboard. Automatically, a popup window appears, as shown in **Figure 9-6**. This window contains all available PHP statements, variables, functions, and other items that begin with the letter “e.”

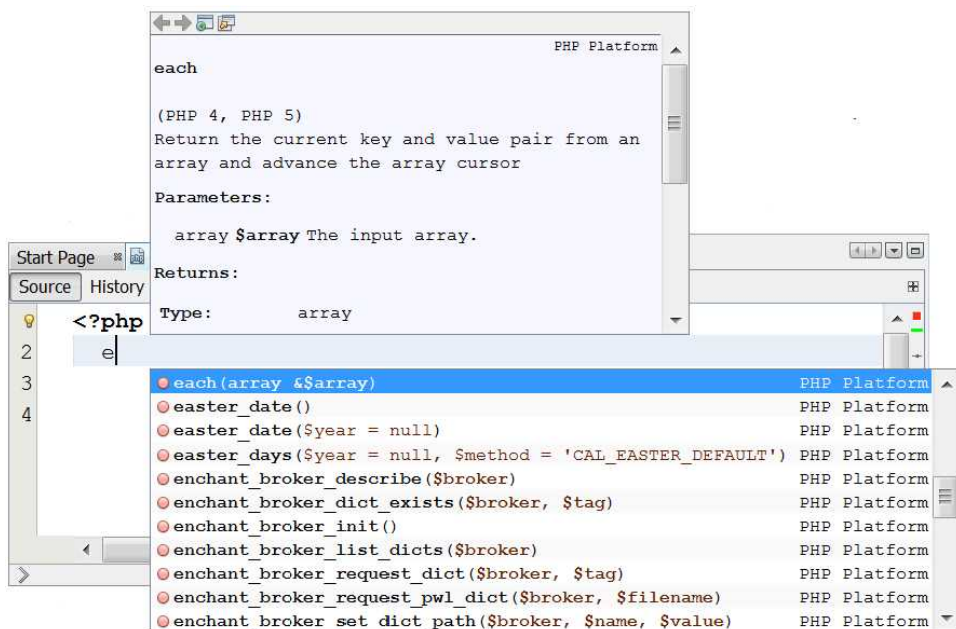


Figure 9-6 The popup screen in the Source Editor window

You can highlight a selection by using the up and down arrow keys on your keyboard. Notice that each time you change a selection, a second window displays a small Help document about the item selected.

Type the second letter “c” from the `echo` statement. Now the options have become fewer as shown here.



Select the option `echo "…;"` and hit “Enter” key. The statement is automatically entered into your script as shown here!



Complete the statement by writing “Hello World” inside double quotes and close the PHP script by writing `?>` at the end of it (as in **Figure 9-7**). Your NetBeans IDE should look like this.

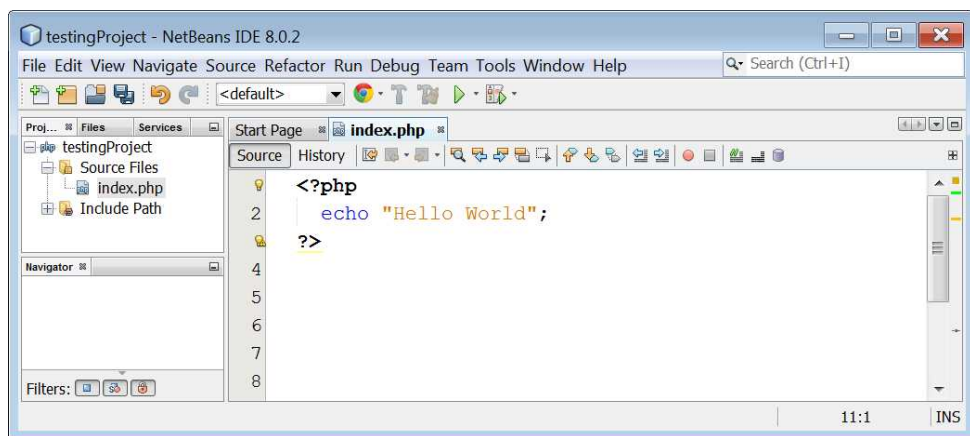


Figure 9-7 Entering a PHP script in the “index.php” file

Now let’s try to execute the script! From the main menu, select “Run→Run File” or hit the SHIFT+F6 key combination. The PHP script executes and the output is displayed in the “Output” window as shown in **Figure 9-8**.

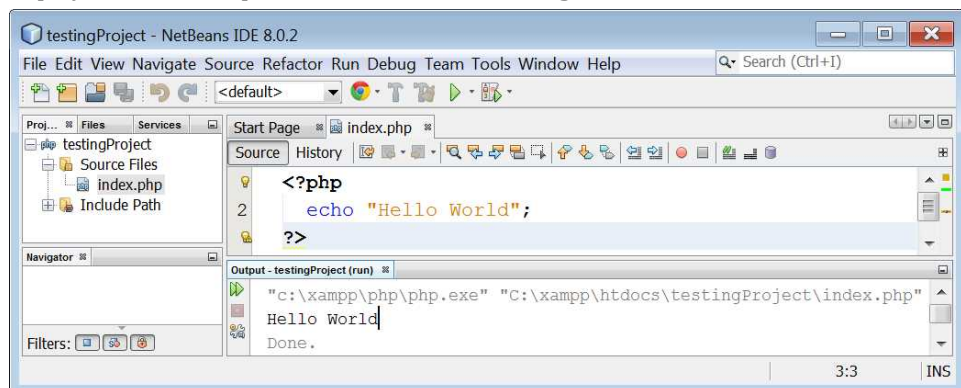


Figure 9-8 Viewing the results of the executed script in the Output window

Congratulations! You have just written and executed your first PHP script!

Now let’s write another PHP script, one that prompts the user to enter his or her name. Type the following PHP statements into the NetBeans IDE and hit SHIFT+F6 to execute the file.

file_9.2.php

```
<?php
echo "Enter your name: ";
$name = trim(fgets(STDIN));
echo "Hello " . $name . "\n";
echo "How do you do?";
?>
```

Remember! You can execute a file by selecting “Run → Run File” from the main menu, or by hitting the **SHIFT+F6** key combination.

Once you execute the script, the message “Enter your name” is displayed in the “Output” window. The script waits for you to enter your name, as shown in **Figure 9-9**.

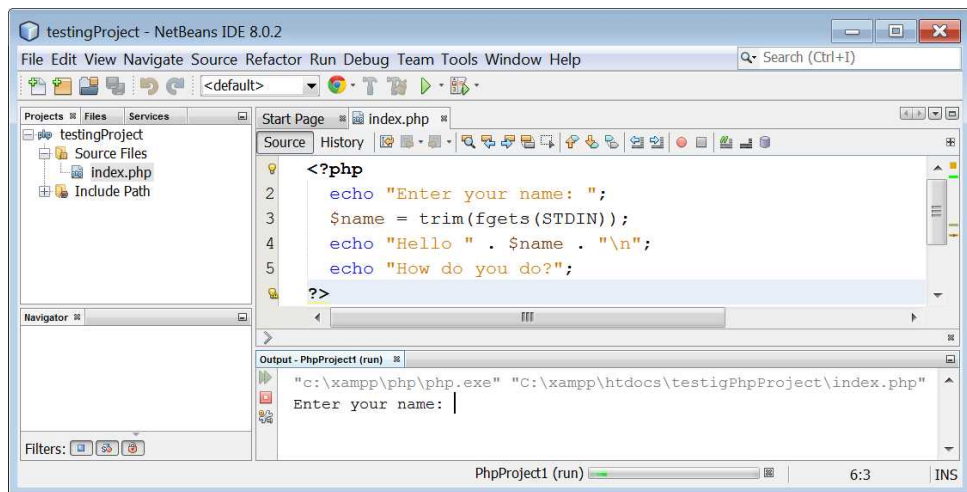


Figure 9-9 Viewing a prompt in the Output window

To enter your name, however, you must place the cursor inside the “Output” window. Then you can type your name and hit the “Enter ↵” key. Once you do that, PHP interpreter continues executing the rest of the statements. When execution finishes, the final output is as shown in **Figure 9-10**.

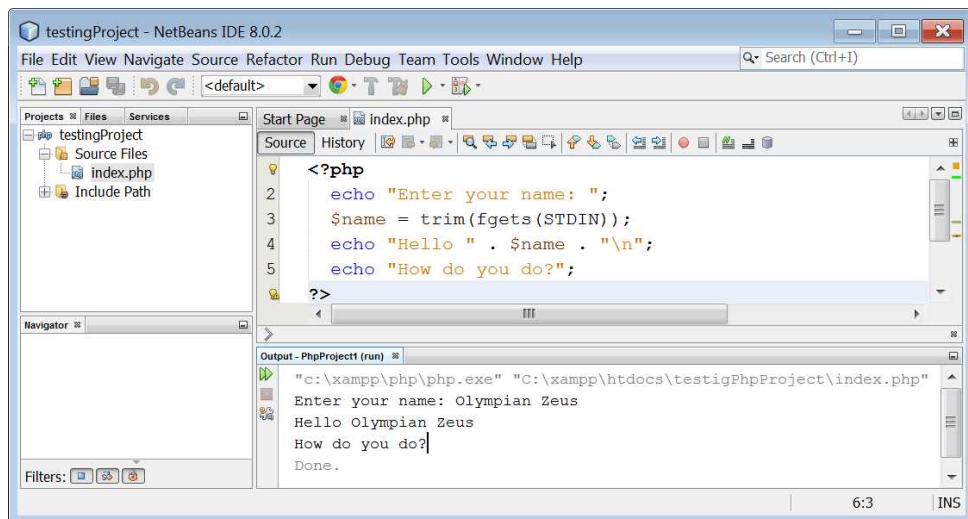


Figure 9-10 Responding to the prompt in the Output window

9.3 What "Debugging" Means

Debugging is the process of finding and reducing the number of defects (bugs) in a computer program, in order to make it perform as expected.

There is a myth about the origin of the term "debugging." In 1940, while Grace Hopper¹ was working on a Mark II Computer at Harvard University, her associates discovered a bug (a moth) stuck in a relay (an electrically operated switch). This bug was blocking the proper operation of the Mark II computer. So, while her associates were trying to remove the bug, Grace Hopper remarked that they were "debugging" the system!

9.4 Debugging PHP Scripts with NetBeans IDE

As you already know, when someone writes code in a high-level language there are two types of errors that he or she might make—syntax errors and logic errors. NetBeans IDE provides all the necessary tools to help you debug your scripts and find the errors.

Debugging syntax errors

Fortunately, NetBeans IDE detects syntax errors while you are typing and underlines them with a wavy red line as shown in **Figure 9-11**.

¹ Grace Murray Hopper (1906–1992) was an American computer scientist and US Navy admiral. She was one of the first programmers of the Harvard Mark I computer, and developed the first compiler for a computer programming language known as A-0 and later a second one, known as B-0 or FLOW-MATIC.

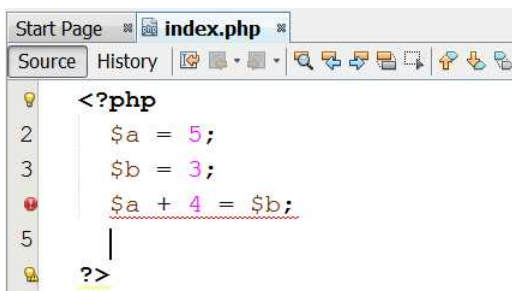


Figure 9-11 In the NetBeans IDE, syntax errors are underlined with a wavy red line

All you have to do is correct the corresponding error and the red line will disappear at once. However, if you are not certain about what is wrong with your code, you can just place your mouse cursor on the red exclamation mark. NetBeans IDE will try to help you by showing a popup window with a brief explanation of the error, as shown in **Figure 9-12**.

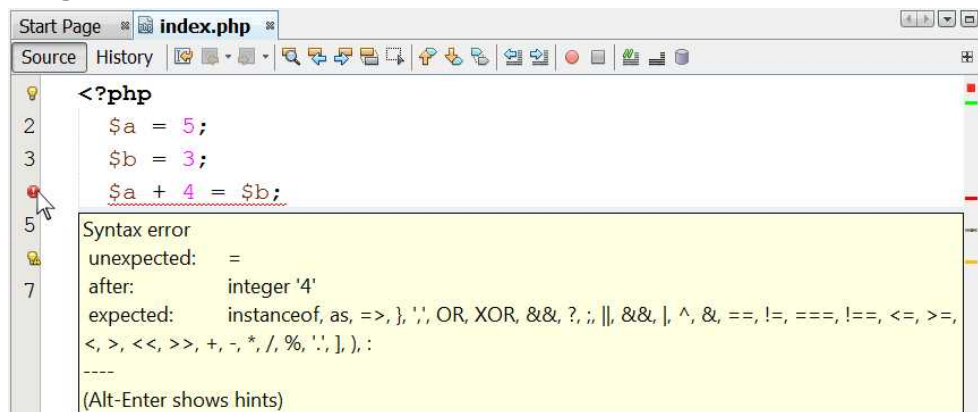


Figure 9-12 The NetBeans IDE shows an explanation of a syntax error

Debugging logic errors by executing scripts step by step

Compared to syntax errors, logic errors are more difficult to find. Since the NetBeans IDE cannot spot and underline logic errors, you are all alone! Let's look at following PHP script, for example. It prompts the user to enter two numbers and calculates and displays their sum. However, it contains a logic error!

file_9.4a.php

```
<?php
$s = 0;
echo "Enter 1st value: ";
$a = trim(fgets(STDIN));
echo "Enter 2nd value: ";
$b = trim(fgets(STDIN));
$$ = $a + $b;
echo "The sum is: ", $s;
```


?>

If you type this script into the NetBeans IDE, you will notice that there is not even one exclamation mark indicating any error. If you execute the script (SHIFT+F6), however, and enter two values, 5 and 2, you can see for yourself that even though the sum of 5 and 2 is 7, NetBeans IDE insists that it is zero, as shown in **Figure 9-13**.

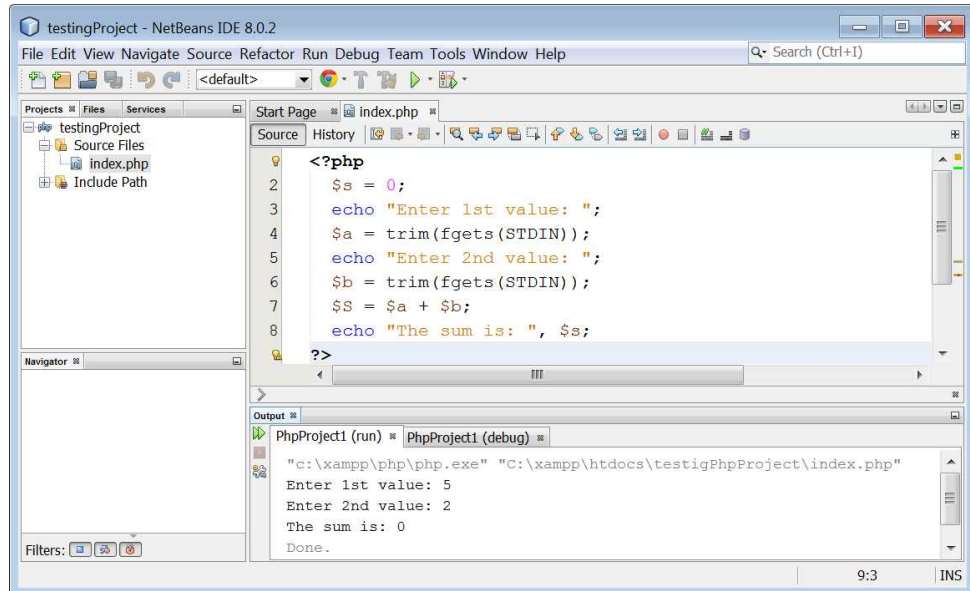


Figure 9-13 Viewing the result of a logic error in the Output window

What the heck is going on? Of course for an expert programmer, correcting this error would be a piece of cake. But for you, a novice one, even though you are trying hard you find nothing wrong. So, where is the error?

Sometimes human eyes get so tired that they cannot see the obvious. So, let's try to use some magic! Let's try to execute the script step by step using the debugger. This gives you the opportunity to observe the flow of execution and take a closer look at the current values of variables in each step.

Start the debugger by selecting "Debug→Debug File" from the main menu or by hitting the CTRL+SHIFT+F5 key combination. By doing this, you enable the debugger and more icons are displayed on the toolbar.



The program counter (the green arrow icon in the gray margin) stops at the first line of the script as shown in **Figure 9-14**.

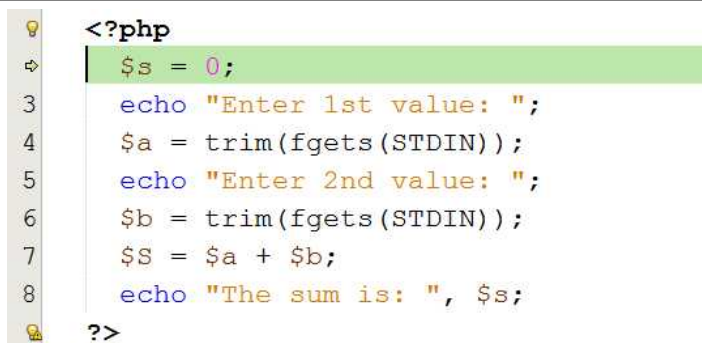


Figure 9-14 Using the debugger in the NetBeans IDE

Notice: The program counter shows you, at any time, which statement is the next to be executed.

Click on the “Step into”  icon on the toolbar. This action tells the PHP interpreter to execute the statement `$s = 0`, and the program counter moves to the second PHP statement, which is the next to be executed.

From the main menu, select “Window→Debugging→Variables”. In the window that appears, you can watch all the variables declared in main memory (RAM) and the value they contain during each step of execution. Until now, the only variable declared and assigned a value is the variable `$s` as shown in **Figure 9-15**.

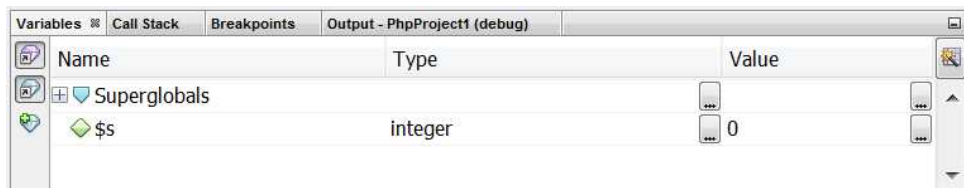


Figure 9-15 The value of variable `$s` is displayed in the Variables window of the debugger

Click on the “Step into”  toolbar icon again. The second statement is executed. You can go back to the “Output” window and see the output result (see **Figure 9-16**).

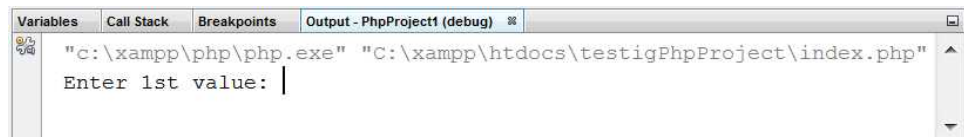


Figure 9-16 Viewing a prompt in the Output window

Click on the “Step into”  toolbar icon again. The third PHP statement is executed and the program waits for you to enter a value. Place the cursor inside the “Output” window, type the value 5, and hit the “Enter ↵” key.

The program counter moves to the fourth PHP statement. If you go back to the “Variables” window you can see that, at the moment, two variables (`$a` and `$s`) are declared in the main memory (RAM), as shown in **Figure 9-17**.

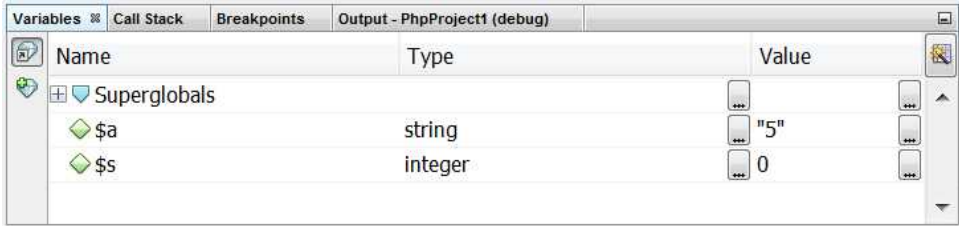


Figure 9-17 The values of variables `$a` and `$s` are displayed in the Variables window

Click on the “Step into”  toolbar icon twice. This action executes the fourth and fifth PHP statements. Place the cursor inside the “Output” window, type the value 2, and hit the “Enter ↵” key.

The program counter moves to the sixth PHP statement. If you go back to the “Variables” window you can see that, at this moment, three variables (`$a`, `$b` and `$s`) are declared in the main memory (RAM) as shown in **Figure 9-18**.

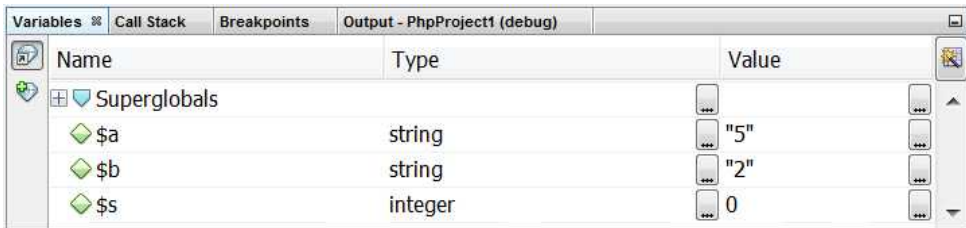


Figure 9-18 The values of variables `$a`, `$b` and `$s` are displayed in the Variables window

Click on the “Step into”  toolbar icon once again. The statement `$S = $a + $b` is executed. What you expected here was the sum of 5 and 2, which is 7, to be assigned to variable `$s`. Instead of this, a new variable `$S` is declared in the main memory (RAM) and the value 7 is assigned to it, as shown in **Figure 9-19**.

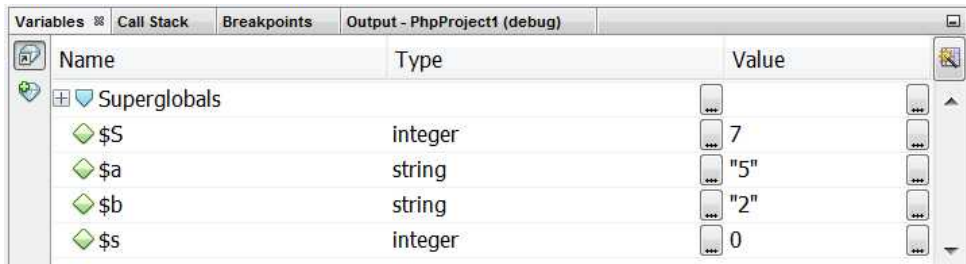


Figure 9-19 All declared variables are displayed in the Variables window of the debugger

Now it becomes more obvious to you! You mistakenly declared two variables in the main memory (RAM), the variable `$s` and the variable `$S` with a capital S. So, when the flow of execution goes to the last statement, `echo "The sum is: ", $s` the value 0 instead of the value 7 is displayed.

You have just found the error! Click on the “Finish Debugger Session”  toolbar icon to cancel execution, correct the error by changing variable `$S` to `$s`, and you are

ready! You just performed your first debugging! Re-execute the script and you will now see that it calculates and displays the sum correctly.

Debugging logic errors by adding breakpoints

Debugging step by step has a big disadvantage. You have to click on the “Step into”  toolbar icon again and again until you reach the position where the error might be. Can you imagine yourself doing this in a large script?

For large scripts there is another approach. If you suspect that the error is somewhere at the end of the script there is no need to debug all of it right from the beginning. You can add a marker (called a “breakpoint”) where you think that the error might be, execute the script and when flow of execution reaches that breakpoint, the flow of execution will pause automatically. You can then take a closer look at the current values of the variables at the position where the script was paused.

Notice: When the script pauses, you have two options for resuming the execution: you can add a second breakpoint somewhere below in the script, click on the “Continue”  toolbar icon, and allow the script to continue execution until that new breakpoint; or, you can just use the “Step into”  toolbar icon and execute the script step by step thereafter.

The next PHP script prompts the user to enter two values and calculates their average value.

However, the script contains a logic error. When the user enters the values 10 and 12, the value 16, instead of 11, is displayed.

file_9.4b.php

```
<?php
echo "Enter 1st value: ";
$a = trim(fgets(STDIN));
echo "Enter 2nd value: ";
$b = trim(fgets(STDIN));
$average = $a + $b / 2;
echo "The average value is: ", $average;
?>
```

You suspect that the problem is somewhere at the end of the script. However, you do not want to debug the entire script, but just the portion in which the error might be. So, let's try to add a breakpoint at the `$average = $a + $b / 2` statement (see **Figure 9-20**). There are two ways to do this: you can click in the left gray margin on the corresponding line number, or you can place the cursor at the line of interest and hit the CTRL+F8 key combination.

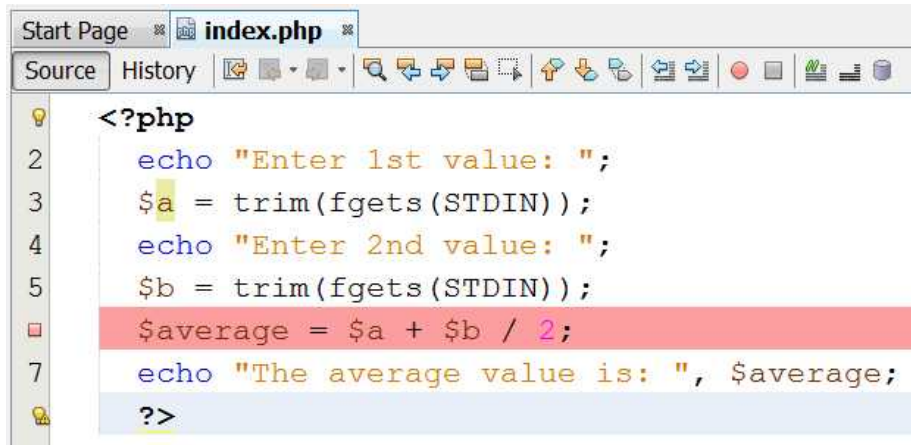


Figure 9-20 Adding a breakpoint to a script

Notice: You know that a breakpoint has been set when the pink rectangle  appears in the gray margin and the corresponding line has pink background highlighting it.

Hit CTRL+SHIFT+F5 to start the debugger. Click on the “Continue”  toolbar icon and enter the values 10 and 12 when requested. You will notice that just after you enter the second number and hit the “Enter ↵” key, the flow of execution pauses at the breakpoint. (You will see that the pink breakpoint highlighting is replaced by the green highlighting of the program counter.)

Remember! You can debug a file by selecting “Debug → Debug File” from the main menu or by hitting the CTRL+SHIFT+F5 key combination.

Now you can take a closer look at the current values of the variables. Variables `$a` and `$b` contain the values 10 and 12 respectively, as they should, so there is nothing wrong with the data input, as shown in **Figure 9-21**.

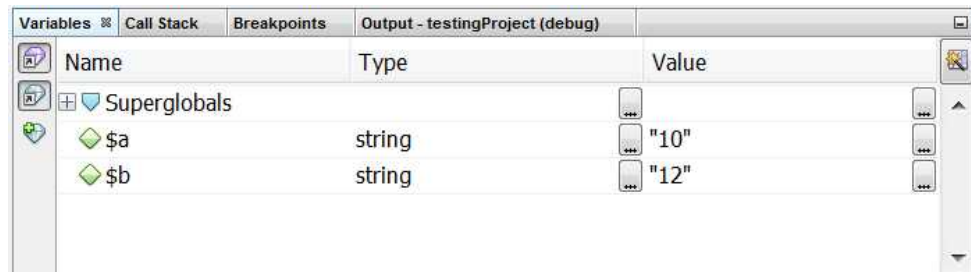


Figure 9-21 Viewing the current values of two variables in the Variables window

Click on the “Step into”  toolbar icon once. The statement `$average = $a + $b / 2` executes and the main memory (RAM) now contains the following values (see **Figure 9-22**).

Variables			Call Stack		Breakpoints		Output - testingProject (debug)	
Name			Type		Value			
+ Superglobals								
◆ \$a			string		"10"			
◆ \$average			integer		16			
◆ \$b			string		"12"			

Figure 9-22 Viewing the current values of three variables in the Variables window

There it is! You just found the statement that erroneously assigns a value of 16, instead of 11, to the variable `$average`! And now comes the difficult part; you should consider why this happens!

After two days of thinking, it becomes obvious! You had just forgotten to enclose `$a + $b` inside parentheses; thus, only the variable `$b` was divided by 2. Click on the “Finish Debugger Session”  toolbar icon, remove all breakpoints, correct the error by enclosing `$a + $b` inside parentheses and you are ready! Re-execute the script and see now that it calculates and displays the average value correctly.

Notice: You can remove a breakpoint the same way you added it: click in the left gray margin on the corresponding line number, or place the cursor at the line that contains a breakpoint and hit the `CTRL+F8` key combination.

9.5 Review Exercises

Complete the following exercises.

1. Type the following PHP script into the NetBeans IDE and execute it step by step. Determine why it doesn't calculate correctly the sum of `1 + 3 + 5`.

```
<?php
    $$$ = 0;
    $S1 = 1;
    $S3 = 3;
    $S5 = 5;

    $S = $S1 + $S3 + $$$;

    echo $S;
?>
```

2. Create a trace table to determine the values of the variables in each step of the PHP script for two different executions. Then, type the script in the NetBeans IDE, execute it step by step, and confirm the results.

The input values for the two executions are: (i) 5, 5; and (ii) 4, 8.

```
<?php
```

```
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
  
$c = $a + $b;  
$d = 5 + $a / $b * $c + 2;  
$e = $c - $d;  
$c -= $d + $c;  
$e--;  
$d -= $c + $a % $c;  
  
echo $c, ", ", $d, ", ", $e;  
?>
```

3. Create a trace table to determine the values of the variables in each step of the PHP script for three different executions. Then, type the script in the NetBeans IDE, execute it step by step, and confirm the results.

The input values for the three executions are: (i) 0.50, (ii) 3, and (iii) 15.

```
<?php  
$b = trim(fgets(STDIN));  
  
$c = 5;  
$c = $c * $b;  
$a = 10 * $c % 10;  
  
echo $a;  
?>
```

Review Questions in “Getting Started with PHP”

Answer the following questions.

1. What is an algorithm?
2. Give the algorithm for making a cup of coffee.
3. What are the five properties of algorithms?
4. Can an algorithm execute forever?
5. What is a computer program?
6. What are the three parties involved in an algorithm?
7. What are the three stages that make up a computer program?
8. Can a computer program be made up of two stages?
9. What is a flowchart?
10. Can a flowchart be entered into a computer as is?
11. What are the basic symbols that flowcharts use?
12. What is meant by the term “reserved words”?
13. What is structured programming?
14. What are the three fundamental control structures of structured programming?
15. Give an example of each control structure using flowcharts.
16. Can a programmer write PHP scripts in a text editor?
17. What is a syntax error? Give one example.
18. What is a logic error? Give one example.
19. What type of error is caused by a misspelled keyword, a missing punctuation character, or the incorrect use of an operator?
20. Why should a programmer add comments in his or her code?
21. Why should a programmer write user-friendly programs/scripts?
22. What does the acronym POLA stand for?
23. What is a variable?
24. How many variables can exist on the left side of the left arrow in flowcharts?
25. In which part of a computer are the values of the variables stored?
26. What is a constant?
27. How can constants be used to help programmers?
28. Why should a computer program avoid division and multiplication operations whenever possible?
29. What are the four fundamental types of variables and constants in PHP?
30. What does the phrase “declare a variable” mean?
31. How do you declare a variable in PHP? Give an example.

32. How do you declare a constant in PHP? Give an example.
33. What symbol is used in flowcharts to display a message?
34. What special sequence of characters is used in PHP to output a “line break”?
35. What symbol is used in flowcharts to let the user enter data?
36. Which symbol is used in PHP as a value assignment operator, and how is it represented in a flowchart?
37. Which arithmetic operators does PHP support?
38. What is a modulus operator, and how is it usually represented in flowcharts?
39. Summarize the rules for the precedence of arithmetic operators.
40. What compound assignment operators does PHP support?
41. What incrementing/decrementing operators does PHP support?
42. What string operators does PHP support?
43. What is a trace table?
44. What are the benefits of using a trace table?
45. Describe the steps involved in swapping the contents (either numeric or alphanumeric) of two variables.
46. Two methods for swapping the values of two variables have been proposed in this book. Which one is better, and why?
47. How many PHP scripts can a PHP project contain?
48. What does the term “debugging” mean?
49. Describe the way in which NetBeans IDE helps you find syntax errors.
50. Describe the ways in which NetBeans IDE helps you find logic errors.

Section 3

Sequence Control Structures

Chapter 10

Introduction to Sequence Control Structures

10.1 What is the Sequence Control Structure?

“Sequence control structure” refers to the line-by-line execution by which statements are executed sequentially, in the same order in which they appear in the script. They might, for example, carry out a series of read or write operations, arithmetic operations, or assignments to variables.

The sequence control structure is the simplest of the three fundamental control structures that you learned about in Chapter 4. The following script shows an example of PHP statements that are executed sequentially.

file_10.1.php

```
<?php
echo "Enter a number: ";
$a = trim(fgets(STDIN));

$b = $a * $a;

echo "The square of ", $a, " is ", $b;
?>
```

Exercise 10.1-1 *Calculating the Area of a Parallelogram*

Write a PHP script that calculates and displays the area of a parallelogram.

Solution

The area of a parallelogram can be calculated using the following formula:

$$\text{Area} = \text{Base} \times \text{Height}$$

In this exercise, the user enters values for *Base* and *Height* and the script calculates and displays the area of the parallelogram. The solution to this problem is shown here.

file_10.1-1.php

```
<?php
echo "Enter the length of Base: ";
$base = trim(fgets(STDIN));
echo "Enter the length of Height: ";
$height = trim(fgets(STDIN));

$area = $base * $height;

echo "The area of the parallelogram is ", $area;
?>
```

Exercise 10.1-2 *Calculating the Area of a Circle*

Write a PHP script that calculates and displays the area of a circle.

Solution

The area of a circle can be calculated using the following formula:

$$\text{Area} = \pi \cdot \text{Radius}^2$$

The value of π is a known quantity, which is 3.14159. Therefore, the only value the user has to enter is a value for *Radius*. The solution to this problem is shown here.

file_10.1-2a.php

```
<?php
    echo "Enter the length of Radius: ";
    $radius = trim(fgets(STDIN));

    $area = 3.14159 * $radius * $radius;

    echo "The area of the circle is ", $area;
?>
```

A much better approach would be with to use a constant, `PI`.

file_10.1-2b.php

```
<?php
    define("PI", 3.14159);

    echo "Enter the length of Radius: ";
    $radius = trim(fgets(STDIN));
    $area = PI * $radius * $radius;
    echo "The area of the circle is ", $area;
?>
```

Exercise 10.1-3 *Calculating Fuel Economy*

In the United States, a car's fuel economy is measured in miles per gallon, or MPG. A car's MPG can be calculated using the following formula:

$$\text{MPG} = \frac{\text{miles driven}}{\text{gallons of gas used}}$$

Write a PHP script that prompts the user to enter the total number of miles he or she has driven and the gallons of gas used. Then the script should calculate and display the car's MPG.

Solution

This is quite a simple case. The user enters the total number of miles he or she has driven and the gallons of gas used, and then the script should calculate and display the car's MPG.

file_10.1-3.php

```
<?php
    echo "Enter miles driven: ";
    $miles_driven = trim(fgets(STDIN));
    echo "Enter gallons of gas used: ";
    $gallons = trim(fgets(STDIN));

    $mpg = $miles_driven / $gallons;

    echo "Your car's MPG is: ", $mpg;
?>
```

Exercise 10.1-4 *Where is the Car? Calculating Distance Traveled*

A car starts from rest and moves with a constant acceleration along a straight horizontal road for a given time. Write a PHP script that prompts the user to enter the acceleration and the time the car traveled, and then calculates and displays the distance traveled. The required formula is

$$S = u_o + \frac{1}{2}at^2$$

where

- **S** is the distance the car traveled, in meters (m)
- **u_o** is the initial velocity (speed) of the car, in meters per second (m/sec)
- **t** is the time the car traveled, in seconds (sec)
- **a** is the acceleration, in meters per second² (m/sec²)

Solution

Since the car starts from rest, the initial velocity (speed) u_o is zero. Thus, the formula becomes

$$S = \frac{1}{2}at^2$$

and the PHP script is

file_10.1-4.php

```
<?php
    echo "Enter acceleration: ";
    $a = trim(fgets(STDIN));
    echo "Enter time traveled: ";
    $t = trim(fgets(STDIN));
```

```

$S = 0.5 * $a * $t * $t;

echo "Your car traveled ", $S, " meters";

?>

```

Exercise 10.1-5 *Kelvin to Fahrenheit*

Write a PHP script that converts a temperature value from degrees Fahrenheit¹ to its degrees Kelvin² equivalent. The required formula is

$$1.8 \times \text{Kelvin} = \text{Fahrenheit} + 459.67$$

Solution

The formula given cannot be used in your script as is. In a computer language such as PHP, it is not permitted to write

```
1.8 * $kelvin = $fahrenheit + 459.67
```

Remember! In the position on the left side of the (=) sign, only one single variable may exist. This variable is actually a region in RAM where a value can be stored.

The script converts degrees Fahrenheit to degrees Kelvin. The value for degrees Fahrenheit is a known value and it is given by the user, whereas the value for degrees Kelvin is what the PHP script should calculate. So, you need to solve for Kelvin. After a bit of work, the formula becomes

$$\text{Kelvin} = \frac{\text{Fahrenheit} + 459.67}{1.8}$$

and the PHP script is shown here.

file_10.1-5.php

```

<?php
echo "Enter a temperature in Fahrenheit: ";
$fahrenheit = trim(fgets(STDIN));

$kelvin = ($fahrenheit + 459.67) / 1.8;

```

¹ Daniel Gabriel Fahrenheit (1686–1736) was a German physicist, engineer, and glass blower who is best known for inventing both the alcohol and the mercury thermometers, and for developing the temperature scale now named after him.

² William Thomson, 1st Baron Kelvin (1824–1907), was an Irish-born British mathematical physicist and engineer. He is widely known for developing the basis of absolute zero (the Kelvin temperature scale), and for this reason a unit of temperature measure is named after him. He discovered the Thomson effect in thermoelectricity and helped develop the second law of thermodynamics.

```
echo "The temperature in Kelvin is ", $kelvin;  
?>
```

Exercise 10.1-6 *Calculating Sales Tax*

An employee needs a script to enter the before-tax price of a product and calculate its final price. Assume a value added tax (VAT) of 19%.

Solution

The sales tax can be easily calculated. You must multiply the before-tax price of the product by the sales tax rate. Be careful—the sales tax is not the final price, but only the tax amount.

The after-tax price can be calculated by adding the initial before-tax price and the sales tax that you calculated beforehand.

In this script you can use a constant named `VAT` for the sales tax rate.

file_10.1-6.php

```
<?php  
define("VAT", 0.19);  
  
echo "Enter the before-tax price of a product: ";  
$price_before_tax = trim(fgets(STDIN));  
  
$sales_tax = $price_before_tax * VAT;  
$price_after_tax = $price_before_tax + $sales_tax;  
  
echo "The after-tax price is: ", $price_after_tax;  
?>
```

Exercise 10.1-7 *Calculating a Sales Discount*

Write a PHP script that prompts the user to enter the price of an item and the discount offered as a percentage (on a scale of 0 to 100). The script should then calculate and display the new price.

Solution

The discount amount can be easily calculated. You must multiply the before-discount price of the product by the discount value and then divide it by 100. The division is necessary since the user enters a value for the discount on a scale of 0 to 100. Be careful—the result is not the final price but only the discount amount.

The final after-discount price can be calculated by subtracting the discount amount that you calculated beforehand from the initial before-discount price.

file_10.1-7.php

```
<?php
echo "Enter the price of a product: ";
$price_before_discount = trim(fgets(STDIN));

echo "Enter the discount offered (0 - 100): ";
$discount = trim(fgets(STDIN));

$discount_amount = $price_before_discount * $discount / 100;
$price_after_discount = $price_before_discount - $discount_amount;

echo "The price after discount is: ", $price_after_discount;
?>
```

Exercise 10.1-8 *Calculating the Sales Tax Rate and Discount*

Write a PHP script that prompts the user to enter the before-tax price of an item and the discount offered as a percentage (on a scale of 0 to 100). The script should then calculate and display the new price. Assume a sales tax rate of 19%.

Solution

This exercise is just a combination of the previous two exercises!

file_10.1-8.php

```
<?php
define("VAT", 0.19);

echo "Enter the price of a product: ";
$price_before_discount = trim(fgets(STDIN));

echo "Enter the discount offered (0 - 100): ";
$discount = trim(fgets(STDIN));

$discount_amount = $price_before_discount * $discount / 100;
$price_after_discount = $price_before_discount - $discount_amount;

$sales_tax = $price_after_discount * VAT;
$price_after_tax = $price_after_discount + $sales_tax;

echo "The discounted after-tax price is: ", $price_after_tax;
?>
```


10.2 Review Exercises

Complete the following exercises.

1. Write a PHP script that prompts the user to enter values for base and height, and then calculates and displays the area of a triangle.
2. Write a PHP script that prompts the user to enter two angles of a triangle, and then calculates and displays the third angle.

Hint: The sum of the measures of the interior angles of any triangle is 180 degrees

3. Write a PHP script that lets a student enter his or her grades from four tests, and then calculates and displays the average grade.
4. Write a PHP script that prompts the user to enter a value for radius, and then calculates and displays the perimeter of a circle.
5. Write a PHP script that prompts the user to enter the charge for a meal in a restaurant, and then calculates and displays the amount of a 10% tip, 7% sales tax, and the total of all three amounts.
6. A car starts from rest and moves with a constant acceleration along a straight horizontal road for a given time in seconds. Write a PHP script that prompts the user to enter the acceleration (in m/sec²) and the time traveled (in sec) and then calculates the distance traveled. The required formula is

$$S = u_o + \frac{1}{2}at^2$$

7. Write a PHP script that prompts the user to enter a temperature in degrees Fahrenheit, and then converts it into its degrees Celsius³ equivalent. The required formula is

$$\frac{C}{5} = \frac{F - 32}{9}$$

8. The Body Mass Index (BMI) is often used to determine whether a person is overweight or underweight for his or her height. The formula used to calculate the BMI is

$$BMI = \frac{weight \cdot 703}{height^2}$$

Write a PHP script that prompts the user to enter his or her weight (in pounds) and height (in inches), and then calculates and displays the user's BMI.

9. Write a PHP script that prompts the user to enter the subtotal and gratuity rate (on a scale of 0 to 100) and then calculates the tip and total. For example if the user enters 30 and 10, the PHP script should display "Tip is \$3.00 and Total is \$33.00".

³ Anders Celsius (1701–1744) was a Swedish astronomer, physicist, and mathematician. He founded the Uppsala Astronomical Observatory in Sweden and proposed the Celsius temperature scale, which takes his name.

10. An employee needs a script to enter the before-tax price of three products and then calculate the final after-tax price of each product, as well as their average value. Assume a value added tax (VAT) of 20%.
11. An employee needs a script to enter the after-tax price of a product, and then calculate its before-tax price. Assume a value added tax (VAT) of 20%.
12. Write a PHP script that prompts the user to enter the initial price of an item and the discount offered as a percentage (on a scale of 0 to 100), and then calculates and displays the final price and the amount of money saved.
13. Write a PHP script that prompts the user to enter the electric meter reading in kilowatt-hours (kWh) at the beginning and end of a month. The script should calculate and display the amount of kWh consumed and the amount of money that must be paid given a cost of each kWh of \$0.06 and a value added tax (VAT) of 20%.
14. Write a PHP script that prompts the user to enter two numbers, which correspond to current month and current day of the month, and then calculates the number of days until the end of the year. Assume that each month has 30 days.

Chapter 11

Manipulating Numbers

11.1 Introduction

Just like every high-level programming language, PHP provides many functions that can be used whenever and wherever you wish. Functions are nothing more than a block of statements packaged as a unit that has a name and performs a specific task.

To better understand functions, let's take Heron's¹ iterative formula that calculates the square root of a positive number.

$$x_{n+1} = \frac{1}{2} \left(x_n + \frac{y}{x_n} \right)$$

where

- y is the number for which you want to find the square root
- x_n is the n -th iteration value of the square root of y

Please don't be disappointed! No one at the present time calculates the square root of a number this way. Fortunately, PHP includes a function for that purpose! This function, which is actually a small subprogram, has been given the name `sqrt` and the only thing you have to do is call it by its name and it will do the job for you. Function `sqrt` probably uses Heron's iterative formula, or perhaps a formula of another ancient or modern mathematician. The truth is that you don't really care! What really matters is that `sqrt` gives you the right result! An example is shown here.

```
$x = trim(fgets(STDIN));  
echo sqrt($x);
```

Even though PHP supports many mathematical functions, this chapter covers only those absolutely necessary for this book's purpose. However, if you need even more information you can visit

<http://www.php.net/manual/en/ref.math.php>

Notice: PHP mathematical functions are used whenever you need to calculate a square root, the sine, the cosine, an absolute value, and so on.

11.2 Useful Mathematical Functions

Absolute value

```
abs (number)
```

This returns the absolute value of *number*.

¹ Heron of Alexandria (c. 10–c. 70 AD) was an ancient Greek mathematician, physicist, astronomer, and engineer. He is considered the greatest experimenter of ancient times. He described the first recorded steam turbine engine, called an "aeolipile" (sometimes called a "Hero engine"). Heron also described a method of iteratively calculating the square root of a positive number. Today, though, he is known best for the proof of "Heron's Formula" which finds the area of a triangle from its side lengths.

Example

file_11.2a.php

```
<?php
    $a = -5;
    $b = abs($a);

    echo abs($a), "\n";           //outputs: 5
    echo $b, "\n";               //outputs: 5
    echo abs(-5.2), "\n";        //outputs: 5.2
    echo abs(5.2), "\n";         //outputs: 5.2
?>
```

Cosine

```
cos (number)
```

This returns the cosine of *number*. The value of *number* must be expressed in radians.

Example

file_11.2b.php

```
<?php
    $p = 3.14159265;
    $a = cos(2 * $p);
    echo $a;                       //outputs: 1
?>
```

Integer value

```
intval (number)
```

This returns the integer value of *number*.

Example

file_11.2c.php

```
<?php
    echo intval(34), "\n";         //outputs: 34
    echo intval(34.9), "\n";       //outputs: 34
    echo intval(-34.999), "\n";    //outputs: -34
?>
```

Pi

```
pi ()
```

This returns the value of π .

Example

file_11.2d.php

```
<?php
```

```
echo pi(); //outputs: 3.1415926535898
?>
```

Power

```
pow (number, exp)
```

This returns the result of *number* raised to the power of *exp*.

Example

file_11.2e.php

```
<?php
$a = 2;
$b = 3;
echo pow($a, $b), "\n"; //outputs: 8

$c = pow(3,2);
echo $c, "\n"; //outputs: 9
?>
```

Notice: Function `pow()` can be used to calculate the square root of a number as well. It is known from mathematics that $\sqrt[n]{X} = X^{\frac{1}{n}}$. So, you can write `$y = pow($x, 1/2)` to calculate the square root of `$x` or you can even write `$y = pow($x, 1/3)` to calculate the cube root of `$x`, and so on!

Random

```
rand (minimum_value, maximum_value)
```

This returns a pseudo-random integer between *minimum_value* and *maximum_value*.

Example

file_11.2f.php

```
<?php
echo rand(2, 10), "\n"; //outputs: 8
echo rand(0, 10)/10, "\n"; //outputs: 0.5
echo rand(20, 25), "\n"; //outputs: 22
?>
```

Notice: Random numbers are widely used in computer games. For example, an “enemy” may show up at a random time or move in random directions. Moreover, random numbers are used in simulation programs, in statistical programs, in computer security to encrypt data, and so on.

Round

```
round (number, [ precision ])
```

This returns the rounded value of *number* to a specified *precision*.

file 11.2g.php

```
<?php
    $a = 5.312;
    $b = 5.315;

    echo round($a, 2), "\n";           //outputs: 5.31
    echo round($b, 2), "\n";           //outputs: 5.32
    echo round(2.3447, 3), "\n";       //outputs: 2.345
?>
```

 $\sin(\text{number})$

Example

file 11.2h.php

```
<?php
    $a = sin(3 * pi() / 2);
    echo $a;                                     //outputs: -1
?>
```

```
sqrt (number)
```

Example

file 11.2i.php

```
<?php
    echo sqrt(2), "\n";           //outputs: 1.4142135623731
    echo sqrt(9), "\n";          //outputs: 3
?>
```

 $\tan(\text{number})$

Example

```
file 11.2j.php
```

[illegible]

Exercise 11.2-1 *Calculating the Distance Between Two Points*

Write a PHP script that prompts the user to enter the coordinates (x, y) of two points and then calculates the straight line distance between them. The required formula is

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Solution

In this exercise, you need to use two functions: the `pow()` function, which returns the result of a number raised to a power; and the `sqrt()` function, which returns the square root of a number.

To simplify things, the terms $(x_1 - x_2)^2$ and $(y_1 - y_2)^2$ are calculated individually and the results are assigned to two temporary variables.

The PHP script is shown here.

file_11.2-1a.php

```
<?php
echo "Enter coordinates for point A: ";
$x1 = trim(fgets(STDIN));
$y1 = trim(fgets(STDIN));

echo "Enter coordinates for point B: ";
$x2 = trim(fgets(STDIN));
$y2 = trim(fgets(STDIN));

$x_temp = pow($x1 - $x2, 2);
$y_temp = pow($y1 - $y2, 2);

$d = sqrt($x_temp + $y_temp);

echo "Distance between points: ", $d;
?>
```

Now let's see another approach.

You should realize that it is actually possible to nest one function within another. Doing that, the result of the nested function is used as an argument for the outer function. This is a writing style that most programmers prefer to follow because it can save a lot of code lines. Of course, if you nest too many functions, no one will be able to understand your code. A nesting of up to four levels is quite acceptable.

The PHP script that uses nested functions is shown here.

file_11.2-1b.php

```
<?php
echo "Enter coordinates for point A: ";
```



```
$x1 = trim(fgets(STDIN));
$y1 = trim(fgets(STDIN));

echo "Enter coordinates for point B: ";
$x2 = trim(fgets(STDIN));
$y2 = trim(fgets(STDIN));

$d = sqrt(pow($x1 - $x2, 2) + pow($y1 - $y2, 2));

echo "Distance between points: ", $d;
?>
```

Exercise 11.2-2 *How Far Did the Car Travel?*

A car starts from rest and moves with a constant acceleration along a straight horizontal road for a given distance. Write a PHP script that prompts the user to enter the acceleration and the distance the car traveled and then calculates the time traveled. The required formula is

$$S = u_o + \frac{1}{2}at^2$$

where

- *S* is the distance the car traveled, in meters (m)
- *u_o* is the initial velocity (speed) of the car, in meters per second (m/sec)
- *t* is the time the car traveled, in seconds (sec)
- *a* is the acceleration, in meters per second² (m/sec²)

Solution

Since the car starts from rest, the initial velocity (speed) *u_o* is zero. Thus, the formula becomes

$$S = \frac{1}{2}at^2$$

Now, if you solve for time, the final formula becomes

$$t = \sqrt{\frac{2S}{a}}$$

In PHP, you can use the `sqrt()` function, which returns the square root of a number.

file_11.2-2.php

```
<?php
echo "Enter acceleration: ";
$a = trim(fgets(STDIN));
echo "Enter distance traveled: ";
```

```
$S = trim(fgets(STDIN));

$t = sqrt(2 * $S / $a);

echo "Your car traveled for ", $t, " seconds";

?>
```

11.3 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Functions are small subprograms that solve small problems.
2. Every programmer must use Heron's iterative formula to calculate the square root of a positive number.
3. The `abs()` function returns the absolute position of an item.
4. The statement `intval(3.59)` returns a result of 3.6.
5. The `pi()` function returns a result of 3.14.
6. The statement `pow(2, 3)` returns a result of 9.
7. The `rand()` function can also return negative random numbers.
8. There is a 50% possibility that the statement `$y = rand(0, 1)` will assign a value of 1 to variable `$y`.
9. The statement `round(3.59, 1)` returns a result of 3.6.
10. To calculate the sinus of 90 degrees, you have to write `$y = sin(pi()/2)`.
11. The statement `$y = sqrt(-2)` is valid.
12. The following code fragment satisfies the property of definiteness.

```
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$x = $a * sqrt($b);
echo $x;
```

11.4 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following calculates the result of the variable `$a` raised to the power of 2?
 - a. `$y = $a * $a;`
 - b. `$y = pow($a, 2);`
 - c. `$y = $a * $a / $a * $a;`
 - d. all of the above
2. What is the value of the variable `$y` when the statement `$y = abs(+5.2)` is executed?
 - a. -5.2

- b. -5
 - c. 0.2
 - d. 5.2
 - e. 5
 - f. none of the above
3. Which of the following calculates the sinus of 180 degrees?
- a. `sin(180)`
 - b. `sin(pi())`
 - c. all of the above
 - d. none of the above
4. What is the value of the variable `$y` when the statement `$y = intval(5/2)` is executed?
- a. 2.5
 - b. 3
 - c. 2
 - d. 0.5
5. What is the value of the variable `$y` when the statement `$y = pow(sqrt(4),2)` is executed?
- a. 4
 - b. 2
 - c. 8
 - d. 16
6. What is the value of the variable `$y` when the statement `$y = round(5.2)/2` is executed?
- a. 2
 - b. 2.5
 - c. 2.6
 - d. none of the above

11.5 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the PHP script for two different executions.

The input values for the two executions are: (i) 9, and (ii) 4.

```
<?php
$a = trim(fgets(STDIN));

$a += 6 / sqrt($a) * 2 + 20;
$b = round($a, 0) % 4;
```

```
$c = $b % 3;

echo $a, ", ", $b, ", ", $c;

?>
```

2. Create a trace table to determine the values of the variables in each step of the PHP script for two different executions.

The input values for the two executions are: (i) -2, and (ii) -3

```
<?php
$a = trim(fgets(STDIN));

$b = abs($a) % 4 + pow($a, 4);
$c = $b % 5;

echo $b, ", ", $c;

?>
```

3. Write a PHP that prompts the user to enter an angle θ in radians and then calculates and displays the angle in degrees. It is given that $2\pi = 360^\circ$.
4. Write a PHP script that calculates the hypotenuse of a right-angled triangle, given its two right angle sides A and B. It is known from the Pythagorean² theorem that

$$\text{hypotenuse} = \sqrt{A^2 + B^2}$$

5. Write a PHP script that prompts the user to enter the angle θ (in degrees) of a right-angled triangle and the length of its adjacent side, and then calculates the length of the opposite side. It is known that $2\pi = 360^\circ$, and

$$\tan(\theta) = \frac{\text{Opposite}}{\text{Adjacent}}$$

² Pythagoras of Samos (c. 571–c. 497 BC) was a famous Greek mathematician, philosopher, and astronomer. He is best known for the proof of the important Pythagorean theorem. He was an influence for Plato. His theories are still used in mathematics today.

Chapter 12

Complex Mathematical Expressions

12.1 Writing Complex Mathematical Expressions

In Chapter 7 you learned all about arithmetic operators but little about how to use them and how to write your own complex mathematical expressions. In this chapter, you are going to learn how easy is to convert mathematical expressions to PHP statements.

Remember! Arithmetic operators follow the same precedence rules as in mathematics, which means that multiplication and division are performed before addition and subtraction. Moreover, when multiplication and division co-exist in the same expression, and since both are of the same precedence, these operations are performed left to right.

Exercise 12.1-1 Representing Mathematical Expressions in PHP

Which of the following PHP statements correctly represent the following mathematical expression?

$$x = \frac{1}{10 + z} 27$$

i. $\$x = 1 * 27 / 10 + \$z$	v. $\$x = (1 / 10 + \$z) * 27$
ii. $\$x = 1 \cdot 27 / (10 + \$z)$	vi. $\$x = 1 / ((10 + \$z) * 27)$
iii. $\$x = 27 / 10 + \$z$	vii. $\$x = 1 / (10 + \$z) * 27$
iv. $\$x = 27 / (10 + \$z)$	viii. $\$x = 1 / (10 + \$z) / 27$

Solution

- i. **Wrong.** Since the multiplication and the division are performed before the addition, this is equivalent to $x = \frac{1}{10} 27 + z$.
- ii. **Wrong.** An asterisk should have been used for multiplication.
- iii. **Wrong.** Since the division is performed before the addition, this is equivalent to $x = \frac{27}{10} + z$.
- iv. **Correct.** This is equivalent to $x = \frac{27}{10 + z}$.
- v. **Wrong.** Inside parentheses, the division is performed before the addition. This is equivalent to $x = \left(\frac{1}{10} + z\right) 27$.
- vi. **Wrong.** Parentheses are executed first and this is equivalent to $x = \frac{1}{(10+z)27}$.
- vii. **Correct.** Division is performed before multiplication (left to right). The term $\frac{1}{10+z}$ is calculated first and then, the result is multiplied by 27.

viii. **Wrong.** This is equivalent to $= \frac{1}{\frac{10+z}{27}}$

Exercise 12.1-2 Writing a Mathematical Expression in PHP

Write a PHP script that calculates the mathematical expression

$$y = 10x - \frac{10 - z}{4}$$

Solution

First, you must distinguish between the data input and the output result. Obviously, the output result is assigned to y and the user must enter values for x and z . The solution for this exercise is shown here.

file_12.1-2.php

```
<?php
echo "Enter value for x: ";
$x = trim(fgets(STDIN));
echo "Enter value for z: ";
$z = trim(fgets(STDIN));

$y = 10 * $x - (10 - $z) / 4;

echo "The result is: ", $y;
?>
```

Exercise 12.1-3 Writing a Complex Mathematical Expression in PHP

Write a PHP script that calculates the mathematical expression

$$y = \frac{5 \frac{3x^2 + 5x + 2}{7w - \frac{1}{z}} - z}{4 \frac{3 + x}{7}}$$

Solution

Oops! Now the expression is more complex! In fact, it is much more complex! So, let's take a look at a quite different approach.

The main idea is to break the complex expression into smaller, simpler expressions and assign each sub-result to temporary variables. In the end, you can build the original expression out of all these temporary variables! This approach is presented here.

file_12.1-3a.php

```
<?php
```

```
echo "Enter value for x: ";
$x = trim(fgets(STDIN));
echo "Enter value for w: ";
$w = trim(fgets(STDIN));
echo "Enter value for z: ";
$z = trim(fgets(STDIN));

$temp1 = 3 * $x * $x + 5 * $x + 2;
$temp2 = 7 * $w - 1 / $z;
$temp3 = (3 + $x) / 7;
$numerator = 5 * $temp1 / $temp2 - $z;
$denominator = 4 * $temp3;

$y = $numerator / $denominator;

echo "The result is: ", $y;

?>
```

You may say, “Okay, but I wasted so many variables and as everybody knows, each variable is a portion of main memory. How can I write the original expression in one single line and waste less memory?”

This job may be a piece of cake for an advanced programmer, but what about you? What about a novice programmer?

The next method will help you write even the most complex mathematical expressions without any syntax or logic errors! The rule is very simple. “After breaking the complex expression into smaller, simpler expressions and assigning each sub-result to temporary variables, start backwards and replace each variable with its assigned expression. Be careful though! When you replace a variable with an expression, you must always enclose the expression in parentheses!”

Confused? Don’t be! It’s easier in action. Let’s try to rewrite the previous PHP script. Starting backwards, replace variables `$numerator` and `$denominator` with their assigned expressions. The result is

```
$y = (5 * $temp1 / $temp2 - $z) / (4 * $temp3);
```

$$\underbrace{(5 * \$temp1 / \$temp2 - \$z)}_{\$numerator} / \underbrace{(4 * \$temp3)}_{\$denominator}$$

Notice: Please note the extra parentheses added.

Now you must replace variables `$temp1`, `$temp2`, and `$temp3` with their assigned expressions, and the one-line expression is complete!

$$\$y = (5 * (3 * \$x * \$x + 5 * \$x + 2) / (7 * \$w - 1 / \$z) - \$z) / (4 * ((3 + \$x) / 7));$$

\$temp1

\$temp2

\$temp3

It may look scary at the end but it wasn't that difficult, was it?
The PHP script can now be rewritten

file_12.1-3b.php

```
<?php
echo "Enter value for x: ";
$x = trim(fgets(STDIN));
echo "Enter value for w: ";
$w = trim(fgets(STDIN));
echo "Enter value for z: ";
$z = trim(fgets(STDIN));

$y = (5 * (3 * $x * $x + 5 * $x + 2) / (7 * $w - 1 / $z) - $z) / (4 * ((3 + $x) / 7));

echo "The result is: ", $y;

?>
```

12.2 Review Exercises

Complete the following exercises.

1. Match each element from the first column with one **or more** elements from the second column.

Expression	Expression
i. 5/pow(\$x, 2)*\$y + pow(\$x,3)	a. 5*\$y/pow(\$x, 2) + pow(\$x, 3)
ii. 5/(pow(\$x, 3)*\$y) + pow(\$x, 2)	b. 5*\$y/\$x*\$x + pow(\$x, 3)
	c. 5/(\$x*\$x*\$x*\$y) + \$x*\$x
	d. 5/(\$x*\$x*\$x)*\$y + \$x*\$x
	e. 5*\$y/(\$x*\$x) + \$x*\$x*\$x
	f. 1/(\$x*\$x*\$x*\$y)*5 + \$x*\$x
	g. \$y/(\$x*\$x)*5 + pow(\$x, 3)
	h. 1/(\$x*\$x)*5*\$y + \$x/\$x*\$x*\$x

2. Write the following mathematical expressions in PHP using one line of code for each.

i. $y = \frac{(x+3)^{5w}}{7(x-4)}$

$$\text{ii. } y = \sqrt[5]{\left(3x^2 - \frac{1}{4}x^3\right)}$$

$$\text{iii. } y = \frac{\sqrt{x^4 - 2x^3 - 7x^2 + x}}{\sqrt[3]{4\left(7x^4 - \frac{3}{4}x^3\right)(7x^2 + x)}}$$

$$\text{iv. } y = \frac{x}{x-3(x-1)} + (x^5\sqrt{x-1})\frac{1}{(x^3-2)(x-1)^3}$$

$$\text{v. } y = \left(\sin\left(\frac{\pi}{3}\right) - \cos\left(\frac{\pi}{2}w\right)\right)^2$$

$$\text{vi. } y = \frac{\left(\sin\left(\frac{\pi}{2}x\right) + \cos\left(\frac{3\pi}{2}w\right)\right)^3}{\left(\tan\left(\frac{2\pi}{3}w\right) - \sin\left(\frac{\pi}{2}x\right)\right)^{\frac{1}{2}}} + 6$$

3. Write a PHP script that prompts the user to enter a value for x and then calculates and displays the result of the following mathematical expression.

$$y = \sqrt{x}(x^3 + x^2)$$

4. Write a PHP script that prompts the user to enter a value for x and then calculates and displays the result of the following mathematical expression.

$$y = \frac{7x}{2x + 4(x^2 + 4)}$$

Suggestion: Try to write the expression in one line of code.

5. Write a PHP script that prompts the user to enter a value for x and w and then calculates and displays the result of the following mathematical expression.

$$y = \frac{x^{x+1}}{\left(\tan\left(\frac{2w}{3}\right) + 5\right) - \tan\left(\frac{x}{2} + 1\right)}^3$$

Suggestion: Try to write the expression in one line of code

6. Write a PHP script that prompts the user to enter a value for x and w and then calculates and displays the result of the following mathematical expression.

$$y = \frac{3 + w}{6x - 7(x + 4)} + (x^5\sqrt{3w + 1})\frac{5x + 4}{(x^3 + 3)(x - 1)^7}$$

Suggestion: Try to write the expression in one line of code.

7. Write a PHP script that prompts the user to enter a value for x and w and then calculates and displays the result of the following mathematical expression.

$$y = \frac{x^x}{\left(\sin\left(\frac{2w}{3}\right) + 5\right) - x}^2 + \frac{(\sin(3x) + w)^{x+1}}{(\sqrt{7w})^{\frac{3}{2}}}$$

Suggestion: Try to write the expression in one line of code

8. Write a PHP script that prompts the user to enter the lengths of all three sides A, B, and C, of a triangle and then calculates and displays the area of the triangle. You can use Heron's formula, which has been known for nearly 2,000 years!

$$Area = \sqrt{S(S - A)(S - B)(S - C)}$$

where S is the semi-perimeter

$$S = \frac{A + B + C}{2}$$

Chapter 13

Exercises with a Quotient and a Remainder

13.1 Introduction

What types of problems might require the use of the quotient and the remainder of an integer division? There is no simple answer to that question! However, quotients and remainders can be used to:

- split a number into individual digits
- examine if a number is odd or even
- convert an elapsed time (in seconds) to hours, minutes, and seconds
- convert an amount of money (in USD) to a number of \$100 notes, \$50 notes, \$20 notes, and such
- calculate the greatest common divisor
- determine if a number is a palindrome
- count the number of digits within a number
- determine how many times a specific digit occurs within a number

Of course, these are some of the uses and certainly you can find so many others. Next you will see some exercises that make use of the quotient and the remainder of integer division.

Exercise 13.1-1 *Calculating the Quotient and Remainder of Integer Division*

Write a PHP script that prompts the user to enter two integers and then calculates the quotient and the remainder of the integer division.

Solution

Since PHP doesn't actually incorporate an arithmetic operator that calculates the integer quotient, you can use the `intval()` function to achieve the same result.

file_13.1-1.php

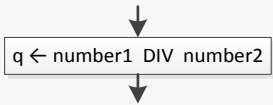
```
<?php
echo "Enter first number: ";
$number1 = trim(fgets(STDIN));

echo "Enter second number: ";
$number2 = trim(fgets(STDIN));

$q = intval($number1 / $number2);
$r = $number1 % $number2;

echo "Integer Quotient: ", $q, "\nInteger Remainder: ", $r;
?>
```

Notice: In flowcharts, in order to calculate the quotient of an integer division, you can use the popular `DIV` operator. An example is shown here.



Exercise 13.1-2 Finding the Sum of Digits

Write a PHP script that prompts the user to enter a four-digit integer and then calculates the sum of its digits.

Solution

What you should keep in mind here is that statements like this one

```
$number = trim(fgets(STDIN));
```

assign the given four-digit integer to one single variable, `$number`, and not to four individual variables.

So, first you must split the integer into its four digits and assign each digit to a separate variable. Then you can calculate the sum of these four variables and get the required result. There are two approaches available.

First Approach

Let's try to understand the first approach using an arithmetic example. Take the number 6753, for example.

First digit = 6	The first digit can be isolated if you divide the given number by 1000 to get the integer quotient <code>\$digit1 = intval(6753 / 1000)</code>
Remaining digits = 753	The remaining digits can be isolated if you divide the given number by 1000 to get the integer remainder <code>\$r = 6753 % 1000</code>
Second digit = 7	The second digit can be isolated if you divide the remaining digits by 100 to get the integer quotient <code>\$digit2 = intval(753 / 100)</code>
Remaining digits = 53	The remaining digits are now <code>\$r = 753 % 100</code>
Third digit = 5	The third digit can be isolated if you divide the remaining digits by 10 to get the integer quotient <code>\$digit3 = intval(53 / 10)</code>

Fourth digit = 3

The last remaining digit, which happens to be the fourth digit, is
 $\$digit4 = 53 \% 10$

The PHP script that solves this algorithm is shown here.

```
file_13.1-2a.php

<?php
echo "Enter a four-digit integer: ";
$number = trim(fgets(STDIN));

$digit1 = intval($number / 1000);
$r = $number % 1000;

$digit2 = intval($r / 100);
$r = $r % 100;

$digit3 = intval($r / 10);
$digit4 = $r % 10;

$sum = $digit1 + $digit2 + $digit3 + $digit4;
echo $sum;

?>
```

The trace table for the script that you have just seen is shown here.

Step	Statement	Notes	\$number	\$digit1	\$digit2	\$digit3	\$digit4	\$r	\$sum
1	echo "Enter a four-digit number: "	The message "Enter a ..." is displayed							
2	\$number = trim(fgets(STDIN))	User enters 6753	6753	?	?	?	?	?	?
3	\$digit1 = intval(\$number / 1000)		6753	6	?	?	?	?	?
4	\$r = \$number % 1000		6753	6	?	?	?	753	?
5	\$digit2 = intval(\$r / 100)		6753	6	7	?	?	753	?
6	\$r = \$r % 100		6753	6	7	?	?	53	?
7	\$digit3 = intval(\$r / 10)		6753	6	7	5	?	53	?

8	\$digit4 = \$r % 10	6753	6	7	5	3	53	?
9	\$sum=\$digit1+\$digit2+\$digit3+\$digit4	6753	6	7	5	3	53	18
10	echo \$sum	Value 18 is displayed						

To further help you, there is also a general purpose PHP script that can be used to split any given integer. Since the length of your script depends on the number of digits, N, all you have to do is write N-1 pairs of statements.

```
<?php
echo "Enter an N-digit integer: ";
$number = trim(fgets(STDIN));

$digit1 = intval($number / 10N-1);
$r = $number % 10N-1;

.
.
.
$digit(N-2) = intval($r / 100);
$r = $r % 100;

$digit(N-1) = intval($r / 10);
$digitN = $r % 10;
```

1st pair of statements

2nd pair of statements

(N-2)th pair of statements

(N-1)th pair of statements

For example, if you want to split a six-digit integer, you need to write five pairs of statements.

file_13.1-2b.php

```
<?php
echo "Enter a six-digit integer: ";
$number = trim(fgets(STDIN));

$digit1 = intval($number / 100000);
$r = $number % 100000;

$digit2 = intval($r / 10000);
$r = $r % 10000;

$digit3 = intval($r / 1000);
$r = $r % 1000;
```

1st pair of statements

2nd pair of statements

3rd pair of statements


```
$digit4 = intval($r / 100);
$r = $r % 100;
```

4th pair of statements

```
$digit5 = intval($r / 10);
$digit6 = $r % 10;
```

5th pair of statements

```
echo $digit1, $digit2, $digit3, $digit4, $digit5, $digit6;
```

```
?>
```

Second Approach

Once more, let's try to understand the second approach using an arithmetic example. Take the same number, 6753, for example.

Fourth digit = 3

The fourth digit can be isolated if you divide the given number by 10 to get the integer remainder

```
$digit1 = 6753 % 10
```

Remaining digits = 675

The remaining digits can be isolated if you divide the given number by 10 to get the integer quotient

```
$r = intval(6753 / 10)
```

Third digit = 5

The third digit can be isolated if you divide the remaining digits by 10 to get the integer remainder

```
$digit3 = 675 % 10
```

Remaining digits = 67

The remaining digits are now

```
$r = intval(675 / 10)
```

Second digit = 7

The second digit can be isolated if you divide the remaining digits by 10 to get the integer remainder

```
$digit3 = 67 % 10
```

First digit = 6

The last remaining digit, which happens to be the first digit, is

```
$digit1 = intval(67 / 10)
```

The PHP script for this algorithm is shown here.

file_13.1-2c.php

```
<?php
```

```
echo "Enter a four-digit integer: ";
```

```
$number = trim(fgets(STDIN));
```

```
$digit4 = $number % 10;
```

```
$r = intval($number / 10);
```

```
$digit3 = $r % 10;
$r = intval($r / 10);

$digit2 = $r % 10;
$digit1 = intval($r / 10);

$sum = $digit1 + $digit2 + $digit3 + $digit4;
echo $sum;

?>
```

To further help you, there is also a general purpose PHP script that can be used to split any given integer. This script uses the second approach. Once again, since the length of your script depends on the number of the digits, N , all you have to do is write $N-1$ pairs of statements.

```
<?php
```

```
echo "Enter a N-digit integer: ";
$number = trim(fgets(STDIN));
```

```
$digit(N) = $number % 10;
$r = intval($number / 10);
```

1st pair of statements

```
$digit(N-1) = $r % 10;
$r = intval($r / 10);
```

2nd pair of statements

```
.
```

```
$digit3 = $r % 10;
$r = intval($r / 10);
```

($N-2$)th pair of statements

```
$digit2 = $r % 10;
$digit1 = intval($r / 10);
```

($N-1$)th pair of statements

For example, if you want to split a five-digit integer, you must use four pairs of statements.

file_13.1-2d.php

```
<?php
```

```
echo "Enter a five-digit integer: ";
$number = trim(fgets(STDIN));
```

1st pair of statements

```
$digit5 = $number % 10;
$r = intval($number / 10);
```

```
$digit4 = $r % 10;
$r = intval($r / 10);
```

```
$digit3 = $r % 10;
$r = intval($r / 10);
```

```
$digit2 = $r % 10;
$digit1 = intval($r / 10);
```

```
echo $digit1, $digit2, $digit3, $digit4, $digit5;
?>
```

2nd pair of statements

3rd pair of statements

4th pair of statements

Exercise 13.1-3 *Displaying an Elapsed Time*

Write a PHP script that prompts the user to enter an integer that represents an elapsed time in seconds and then displays it in the format “DD days HH hours MM minutes and SS seconds”. For example if the user enters the number 700005, the message “8 days 2 hours 26 minutes and 45 seconds” must be displayed.

Solution

As you may already know, there are 60 seconds in a minute, 3600 seconds in an hour, and 86400 seconds in a day.

Let’s try to analyze number 700005 using the first approach.

Days = 8	The number of days can be isolated if you divide the given integer by 86400 to get the integer quotient <code>\$days = intval(700005 / 86400);</code>
Remaining seconds = 8805	The remaining seconds can be isolated if you divide the given integer by 86400 to get the integer remainder <code>\$r = 700005 % 86400;</code>
Hours = 2	The number of hours can be isolated if you divide the remaining seconds by 3600 to get the integer quotient <code>\$hours = intval(8805 / 3600);</code>
Remaining seconds = 1605	The remaining seconds are now <code>\$r = 8805 % 3600;</code>

Minutes = 26

The number of minutes can be isolated if you divide the remaining seconds by 60 to get the integer quotient

```
$minutes = intval(1605 / 60);
```

Seconds = 45

The last remainder, which happens to be the number of seconds left, is

```
$seconds = 1605 % 60;
```

The PHP script for this algorithm is shown here.

file_13.1-3.php

```
<?php
echo "Enter a period of time in seconds: ";
$number = trim(fgets(STDIN));

$days = intval($number / 86400);           // 60 * 60 * 24 = 86400
$r = $number % 86400;

$hours = intval($r / 3600);                 // 60 * 60 = 3600
$r = $r % 3600;

$minutes = intval($r / 60);
$seconds = $r % 60;

echo $days, " days ", $hours, " hours ";
echo $minutes, " minutes and ", $seconds, " seconds";
?>
```

Exercise 13.1-4 *Reversing a Number*

Write a PHP script that prompts the user to enter a three-digit integer and then reverses it. For example, if the user enters the number 375, the number 573 should be displayed.

Solution

To isolate the three digits of the given number, you can use either first or second approach. Afterward, the only difficulty in this exercise is to build the reversed number. Take the number 375, for example. The three digits, after isolation, are

```
$digit1 = 3;
$digit2 = 7;
$digit3 = 5;
```

You can build the reversed number by simply calculating the sum of the products:

$$\text{digit3} \times 100 + \text{digit2} \times 10 + \text{digit1} \times 1$$

For a change, you can split the given number using the second approach. The PHP script will look like this.

file_13.1-4.php

```
<?php
echo "Enter a three-digit integer: ";
$number = trim(fgets(STDIN));

$digit3 = $number % 10;           //This is the rightmost digit
$r = intval($number / 10);

$digit2 = $r % 10;                //This is the digit in the middle
$digit1 = intval($r / 10);        //This is the leftmost digit

$reversed = $digit3 * 100 + $digit2 * 10 + $digit1;
echo $reversed;
?>
```

13.2 Review Exercises

Complete the following exercises.

1. Write a PHP script that prompts the user to enter any integer and then multiplies its last digit by 8 and displays the result.
Hint: You can isolate the last digit of any integer using a modulus 10 operation.
2. Write a PHP script that prompts the user to enter a five-digit integer and then reverses it. For example, if the user enters the number 32675, the number 57623 must be displayed.
3. Write a PHP script that prompts the user to enter an integer and then it displays 1 when the number is odd; otherwise, it displays 0. Try not to use any decision control structures since you haven't learned anything about them yet!
4. Write a PHP script that prompts the user to enter an integer and then it displays 1 when the number is even; otherwise, it displays 0. Try not to use any decision control structures since you haven't learned anything about them yet!
5. Write a PHP script that prompts the user to enter an integer representing an elapsed time in seconds and then displays it in the format "WW weeks DD days HH hours MM minutes and SS seconds." For example, if the user enters the number 2000000, the message "3 weeks 23 days 3 hours 33 minutes and 20 seconds" should be displayed.
6. Inside an ATM bank machine there are notes of \$20, \$10, \$5, and \$1. Write a PHP script that prompts the user to enter the amount of money he or she

wants to withdraw (using an integer value) and then displays the least number of notes the ATM should give. For example, if the user enters an amount of \$76, the script should display the message “3 notes of \$20, 1 note of \$10, 1 note of \$5, and 1 note of \$1”.

7. A robot arrives on the moon in order to perform some experiments. Each of the robot's steps is 25 inches long. Write a PHP script that prompts the user to enter the number of steps the robot made and then calculates and displays the distance traveled in miles, feet, yards, and inches. For example, if the distance traveled is 100000 inches, the script should display the message “1 mile, 1017 yards, 2 feet, and 4 inches”.

It is given that

- 1 mile = 63360 inches
- 1 yard = 36 inches
- 1 foot = 12 inches

Chapter 14

Manipulating Strings

14.1 Introduction

Generally speaking, a string is anything that you can type using the keyboard, including letters, symbols (such as &, *, and @), and digits. Sometimes a program deals with data that comes in the form of strings (text). In PHP, a string is always enclosed in double quotes like this: " ".

Notice: Even though PHP permits strings to be enclosed in single quotes as well, this book prefers the use of double quotes. There is indeed a small difference between them but it's nothing really special that you must know right now.

Each statement in the next example outputs a string.

```
<?php
    echo "Everything enclosed in double quotes is a";
    echo "string, even the numbers below:";
    echo "3, 4, 7";
    echo "\nYou can even mix letters, symbols and numbers like this:";
    echo "The result of 3 + 4 equals to 4";
?>
```

Strings are everywhere—from word processors, to web browsers, to text messaging programs. Many exercises in this book actually make extensive use of strings. Even though PHP supports many useful functions for manipulating strings, this chapter covers only those functions that are absolutely necessary for this book's purpose. However, if you need even more information you can visit

<http://www.php.net/manual/en/ref.strings.php>

Notice: PHP string functions can be used when there is a need, for example, to isolate a number of characters from a string, to remove spaces that might exist at the beginning of it, or to convert all of its characters to uppercase.

Remember! Functions are nothing more than small subprograms that solve small problems. A subprogram can be defined as a block of statements packaged as a unit that has a name and performs a specific task.

14.2 The Position of a Character in a String

Let's use the text «Hello World» for the following example. The string consists of 11 characters (including the space character between the two words). The position of each character is shown here.

0	1	2	3	4	5	6	7	8	9	10
H	e	l	l	o		W	o	r	l	d

PHP numerates characters assuming that the first one is at position 0, the second one is at position 1, and so on. Please note that the space between the two words is considered a character as well.

Remember! A space is a character just like any other character. Just because nobody can see it, it doesn't mean it doesn't exist!

14.3 Retrieving an Individual Character From a String

PHP allows you to retrieve the individual characters of a string using substring notation. You can use index 0 to access the first character, index 1 to access the second character, and so on. The index of the last character is 1 less than the length of the string. The following PHP script shows an example.

file_14.3.php

```
<?php
    $a = "Hello World";

    echo $a[0];           //it displays the first letter
    echo $a[6];           //it displays the letter W
    echo $a[10];          //it displays the last letter
?>
```

Notice: Please note that the space between the words “Hello” and “World” is considered a character as well. So, the letter *W* exists in position 6 and not in position 5.

If you attempt to use an invalid index such as a negative one or an index greater than the length of the string, PHP displays an error message as shown in **Figure 14-1**.

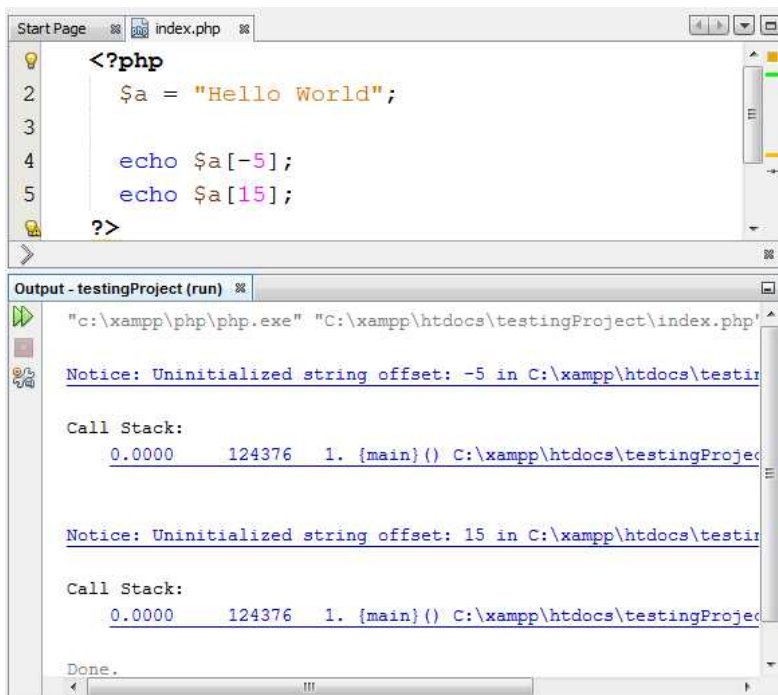


Figure 14-1 An error message indicating an invalid index

Remember! String indexes must be in a range from 0 to one less than the length of the string.

Exercise 14.3-1 *Displaying a String Backwards*

Write a PHP script that prompts the user to enter any string with five letters and then displays its contents backwards. For example, if the string entered is “Hello”, the script should display “olleH”.

Solution

Let’s say that user’s input is assigned to variable `$str`. You can access the fifth letter using `$str[4]`, the fourth letter using `$str[3]`, and so on.

The PHP script is shown here. A period character is used to reassemble the final reversed string.

file_14.3-1.php

```
<?php
echo "Enter a word with five letters: ";
$str = trim(fgets(STDIN));

echo $str[4] . $str[3] . $str[2] . $str[1] . $str[0];
?>
```

14.4 Useful String Functions

Trimming

Trimming is the process of removing whitespace characters from the beginning or the end of a string.

Some of the whitespace characters that are removed with the trimming process are:

- " " an ordinary space
- "\t" a tab
- "\n" a new line (line feed)
- "\r" a carriage return

For example, you can trim any spaces that the user mistakenly entered at the end of a string.

There are actually three functions that can help you trim a string.

Left trimming

```
ltrim (string)
```

This removes any whitespace characters from the beginning of *string*.

Example

file_14.4a.php

```
<?php
```

```
$a = "           Zeus!";

$b = ltrim($a);

echo "Hello ", $b;           //Outputs: Hello Zeus!
?>
```

Right trimming

```
rtrim (string)
```

This removes any whitespace characters from the end of *string*.

Example

file_14.4b.php

```
<?php
$a = "Hello           ";

$b = rtrim($a);

echo $b, " Hera!";           //Outputs: Hello Hera!
?>
```

Left & Right trimming

```
trim (string)
```

This removes any whitespace characters from both the beginning and the end of *string*.

Example

file_14.4c.php

```
<?php
$a = "      Hello      ";

$b = trim($a);

echo $b, " Poseidon!";       //Outputs: Hello Poseidon!
?>
```

String replacement

```
str_replace (search, replace, subject)
```

This searches in *subject* and replaces all occurrences of the *search* string with the *replace* string.

Example

file_14.4d.php

```
<?php
```

```
$a = "I am newbie in C++. C++ rocks!";

$b = str_replace("C++", "PHP", $a);

echo $b; //Outputs: I am newbie in PHP. PHP rocks
?>
```

Counting the number of characters

```
strlen (string)
```

This returns the length of *string* or, in other words, the number of characters *string* consists of (including space characters, symbols, numbers, and so on).

Example

file_14.4e.php

```
<?php
$a = "Hello Olympians!";

echo strlen($a); //Outputs: 16
echo "\n";
echo strlen("I am newbie in PHP"); //Outputs: 18
?>
```

Finding string position

```
strpos (subject, search)
```

This finds the numeric position of the first occurrence of *search* in the *subject* string.

Example

file_14.4f.php

```
<?php
$a = "I am newbie in PHP. PHP rocks!";

$i = strpos($a, "newbie");

echo $i; //Outputs: 5
?>
```

Remember! The first character is at position 0.

Converting to lowercase or uppercase

There are two functions that let you convert all letters in a string to lowercase or uppercase.

To lowercase

```
strtolower (string)
```

This returns the *string*, converted to lowercase.

Example

file_14.4g.php

```
<?php
    $a = "My NaMe is JohN";

    $b = strtolower($a);

    echo $b; //Outputs: my name is john
?>
```

To uppercase

```
strtoupper (string)
```

This returns the *string*, converted to uppercase.

Example

file_14.4h.php

```
<?php
    $a = "My NaMe is JohN";

    $b = strtoupper($a);

    echo $b; //Outputs: MY NAME IS JOHN
?>
```

Getting part of a string

```
substr (string, start [, length])
```

This returns a portion of *string*. Specifically, it returns *length* number of characters starting from *start* position. The *length* argument is optional. If it is omitted, the substring starting from *start* position until the end of *string* is returned.

Example

file_14.4i.php

```
<?php
    $a = "Hello Athena";

    echo substr ($a, 6, 3);           //Outputs: Ath
    echo "\n";
    echo substr ($a, 7);             //Outputs: thena
?>
```

Checking if string is numeric

```
is_numeric (string)
```

This checks if *string* is numeric and returns true or false accordingly. Numeric strings consist of an optional sign, any number of digits, an optional decimal part, and an optional exponential part.

Example

file_14.4j.php

```
<?php
    $a = "753";
    echo is_numeric($a), "\n";           //Value 1 (true) is displayed

    $a = "753.6";
    echo is_numeric($a), "\n";           //Value 1 (true) is displayed

    $a = "-10E-3";
    echo is_numeric($a), "\n";           //Value 1 (true) is displayed

    $a = "Hello";
    echo is_numeric($a), "\n";           //This displays nothing (false)
?>
```

Exercise 14.4-1 Switching the Order of Names

Write a PHP script that prompts the user to enter in one single string both first and last name. In the end, the script should change the order of the two names.

Solution

This exercise is not the same as the one that you learned in Chapter 8, which swapped the numeric values of two variables. In this exercise, you must split the string and assign each name to a different variable. If you manage to do so, then you can just rejoin them in a different order.

Let’s try to understand this exercise using an example. The string that you must split and the position of its individual character is shown here.

0	1	2	3	4	5	6	7	8
T	o	m		S	m	i	t	h

The character that visually separates the first name from the last name is the space character between them. The problem is that this character is not always at position 3. Someone can have a short first name like “Tom” and someone else can have a longer one like “Robert”. Thus, you need something that actually finds the position of the space character regardless of the content of the string.

Function `strpos()` is what you are looking for! If you use it to find the position of the space character in the string “Tom Smith”, it returns the value 3. But if you use it to find the space character in another string, such as “Angelina Brown”, it returns the value 8 instead.

Notice: *The value 3 is not just the position where the space character exists. It also represents the number of characters that the word “Tom” contains! The same applies to the value 8 that is returned for the string “Angelina Brown”. It represents both the position where the space character exists and the number of characters that the word “Angelina” contains!*

The PHP script for this algorithm is shown here.

file_14.4-1.php

```
<?php
echo "Enter your full name: ";
$full_name = trim(fgets(STDIN));

//find the position of space character. This is also the number
//of characters first name contains
$space_pos = strpos($full_name, " ");

//get $space_pos number of characters starting from position 0
$name1 = substr($full_name, 0, $space_pos);

//get the rest of the characters starting from position $space_pos + 1
$name2 = substr($full_name, $space_pos + 1);

$full_name = $name2 . " " . $name1;

echo $full_name;
?>
```

Notice: *Please note that this script cannot be applied to a Spanish name such as “María Teresa García Ramírez de Arroyo.” The reason is obvious!*

Exercise 14.4-2 Creating a Login ID

Write a PHP script that prompts the user to enter his or her last name and then creates a login ID from the first four letters of the name (in lowercase) and a three-digit random integer.

Solution

To create a random integer you can use the `rand()` function. Since you need a random integer of three digits, the range must be between 100 and 999.

The PHP script for this algorithm is shown here.

file_14.4-2.php

```
<?php
    echo "Enter last name: ";
    $last_name = trim(fgets(STDIN));

    $random_int = rand(100, 999);

    $login_ID = strtolower( substr($last_name, 0, 4) ) . $random_int;
    echo $login_ID;
?>
```

Notice: Please note how the function `substr()` is nested within the function `strtolower()`. The result of the inner (nested) function is used as an argument for the outer function. This is a writing style that most programmers prefer to follow because it helps to save a lot of code lines. Of course, if you nest too many functions, no one will be able to understand your code. A nesting of up to four levels is quite acceptable.

Exercise 14.4-3 *Creating a Random Word*

Write a PHP script that creates a random word consisting of five letters.

Solution

To create a random word you need a string that contains all 26 letters of the English alphabet. Then you can use the `rand()` function to choose a random letter between position 0 and 25.

The PHP script for this algorithm is shown here.

file_14.4-3.php

```
<?php
    $alphabet = "abcdefghijklmnopqrstuvwxyz";

    $random_word = $alphabet[rand(0, 25)] .
                   $alphabet[rand(0, 25)] .
                   $alphabet[rand(0, 25)] .
                   $alphabet[rand(0, 25)] .
                   $alphabet[rand(0, 25)] ;

    echo $random_word;
?>
```


14.5 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A string is anything that you can type using the keyboard.
2. Strings must be enclosed in parentheses.
3. The phrase "Hi there!" contains 8 characters.
4. In the phrase "Hi there!" the letter "t" is at position 3.
5. The statement `$y = $a[1]` assigns the second letter of the string contained in variable `$a` to variable `$y`.
6. The statement `$y = $a[-1];` is a valid statement.
7. Trimming is the process of removing whitespace characters from the beginning or the end of a string.
8. The statement `$y = trim("Hello Aphrodite")` assigns the value "HelloAphrodite" to variable `$y`.
9. The statement `echo str_replace("Hi", "Hello", "Hi there! ")` displays the message "Hello there!".
10. The statement `$index = strpos("Hi there", "the")` assigns the value 4 to the variable `$index`.
11. The statement `echo strtoupper("hello there!")` displays the message "Hello There".
12. The statement `echo substr("Hello there!", 0)` displays the message "Hello there!".
13. The statement `echo substr($a, 0, strlen($a))` displays some letters of the variable `$a`.
14. The statement `echo substr($a, strlen($a) - 1, 1)` is equivalent to the statement `echo $a[strlen($a) - 1]`.
15. The statement `echo strtoupper(substr("hello there!", 0, 5))` displays the word "HELLO".
16. The statement `is_numeric("Exercise 1")` returns true.
17. The statement `is_numeric("10E-2")` returns true.

14.6 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. Which of the following is **not** a string?
 - a. "Hello there!"
 - b. "13"
 - c. "13.5"
 - d. All of the above are strings.
2. In which position does the space character in the string "Hello Zeus!", exist?
 - a. 6

- b. 5
 - c. Space is not a character.
 - d. none of the above
- 3. The statement `echo substr($a, strlen($a) - 1, 1)` displays
 - a. the last character of variable `$a`.
 - b. the second to last character of variable `$a`.
 - c. The statement is not valid.
- 4. The statement `ltrim(rtrim($a))` is equivalent to the statement
 - a. `rtrim(ltrim($a))`.
 - b. `trim($a)`.
 - c. `trim(rtrim(ltrim($a)))`.
 - d. all of the above
- 5. The statement `str_replace($a, " ", "")`
 - a. adds a space between each letter in the variable `$a`.
 - b. removes all space characters from the variable `$a`.
 - c. empties the variable `$a`.
- 6. The statement `str_replace(" Hello ", " ", "")` is equivalent to the statement
 - a. `str_replace(" Hello ", "", " ")`.
 - b. `trim(" Hello ")`.
 - c. `rtrim(" Hello ")`.
 - d. all of the above
 - e. none of the above
- 7. The statement `echo strlen("")` displays
 - a. nothing.
 - b. 1.
 - c. 0.
 - d. The statement is invalid.
 - e. none of the above
- 8. Which value assigns the statement `$Shakespeare = strpos("2b || !2b", "b")` to the variable `$Shakespeare`?
 - a. 1
 - b. 2
 - c. 6
 - d. none of the above

9. What does the following code fragment?

```
$a = "Hi there";  
$b = substr($a, strpos($a, " ") + 1);
```

- a. It assigns the word “Hi” to the variable \$b.
 - b. It assigns a space character to the variable \$b.
 - c. It assigns the word “there” to the variable \$b.
 - d. none of the above
10. In which case(s) does function `is_numeric()` return true?
- a. `is_numeric("1E21")`
 - b. `is_numeric("21")`
 - c. `is_numeric("1E-21")`
 - d. all of the above

14.7 Review Exercises

Complete the following exercises.

1. Write a PHP script that prompts the user to enter his or her first name, middle name, last name, and his or her preferred title (Mr., Mrs., Ms., Dr., and so on) and displays them formatted in all the following ways.

Title FirstName MiddleName LastName

FirstName MiddleName LastName

LastName, FirstName

LastName, FirstName MiddleName

LastName, FirstName MiddleName, Title

FirstName LastName

For example, assume that the user enters the following:

First name: Aphrodite

Middle name: Maria

Last name: Boura

Title: Ms.

The script should display the user’s name formatted in all the following ways:

Ms. Aphrodite Maria Boura

Aphrodite Maria Boura

Boura, Aphrodite

Boura, Aphrodite Maria

Boura, Aphrodite Maria, Ms.

Aphrodite Boura

2. Write a PHP script that creates and displays a random word consisting of five letters. The first letter must be a capital letter.
3. Write a PHP script that prompts the user to enter his or her name and then creates a secret password consisting of three letters (in lowercase) randomly picked up from his or her name, and a random four-digit number. For example, if the user enters “Vassilis Bouras” a secret password can probably be one of “sar1359” or “vbs7281” or “bor1459”.

Review Questions in “Sequence Control Structures”

Answer the following questions.

1. What is a sequence control structure?
2. What operations can a sequence control structure perform?
3. Give some examples of how you can use the quotient and the remainder of an integer division.
4. What is a function?
5. What does the term “nest a function” mean?

Section 4

Decision Control Structures

Chapter 15

Introduction to Decision Control Structures

15.1 What is a Decision Control Structure?

All you have learned so far is just the sequence control structure, where statements are executed sequentially, in the same order in which they appear in the script. However, in serious PHP programming, rarely do you want the statements to be executed sequentially. Many times you want a block of statements to be executed in one situation and an entirely different block of statements to be executed in another situation. This is where a decision control structure can take action!

A decision control structure evaluates a Boolean expression or a set of Boolean expressions and then decides which block of statements to execute.

15.2 What is a Boolean Expression?

A Boolean expression is an expression that results in a Boolean value, that is, either true or false.

15.3 How to Write Boolean Expressions

A simple Boolean expression is written as

Operand1 **Comparison Operator** *Operand2*

where

- *Operand1* and *Operand2* can be values, variables or mathematical expressions
- *Comparison Operator* can be one of those shown in **Table 15-1**

Table 15-1 Comparison Operators in PHP

Comparison Operator	Description
==	Equal (not assignment)
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Notice: A very common mistake that novice programmers make when writing PHP scripts is to confuse the assignment operator with the equal operator. They frequently make the mistake of writing `$x = 5` when they actually want to say `$x == 5`.

Here are some examples of Boolean expressions:

- `$x == 5`. This can be read as “test if `$x` is equal to 5”
- `$x > $y`. This can be read as “test if `$x` is greater than `$y`”
- `$x <= $y`. This can be read as “test if `$x` is less than or equal to `$y`”
- `$x != 3 * $y + 4`. This can be read as “test if `$x` is not equal to the result of the expression `3 * $y + 4`”

Notice: For humans, Boolean expressions should be interpreted as questions. They should be read as “Is something equal to/greater than/less than something else?” and the answer is just a “Yes” or a “No” (true or false).

Moreover, given that a Boolean expression actually returns a value (true or false), this value can be directly assigned to a variable. For example, the expression

```
$a = $x > $y;
```

assigns a value of true or false to variable \$a. It can be read as “If the content of variable \$x is greater than the content of variable \$y, assign the value true to variable \$a; otherwise, assign the value false.” This next example displays the number 1 on the screen.

```
<?php
    $x = 8;
    $y = 5;
    $a = $x > $y;

    echo $a;
?>
```

Notice: Please note that in PHP, the Boolean value true can also be represented by the value of 1.

Exercise 15.3-1 *Filling in the Table*

Fill in the following table with the words “True” or “False” according to the values of the variables \$a, \$b, and \$c.

\$a	\$b	\$c	\$a == 10	\$b <= \$a	\$c > 3 * \$a - \$b
3	-5	7			
10	10	21			
-4	-2	-9			

Solution

Some notes about the table:

- The Boolean expression `$a == 10` is true when the content of the variable \$a is 10.
- The Boolean expression `$b <= $a` is true when \$b is less than or equal to \$a.
- The Boolean expression `$c > 3 * $a - $b` is true when \$c is greater than the result of the expression `3 * $a - $b`. Please be careful with the cases where \$b is negative. For example, in the first line, \$a is equal to 3 and \$b is equal to

-5. The result of the expression `3 * $a - $b` is `3 * 3 - (-5) = 3 * 3 + 5 = 14`.

So, the table becomes

\$a	\$b	\$c	\$a == 10	\$b <= \$a	\$c > 3 * \$a - \$b
3	-5	7	False	True	False
10	10	21	True	True	True
-4	-2	-9	False	False	True

15.4 Logical Operators and Complex Boolean Expressions

A more complex Boolean expression can be built of simpler Boolean expressions and can be written as

`Boolean_Expression1` **Logical Operator** `Boolean_Expression2`

where

- `Boolean_Expression1` and `Boolean_Expression2` can be any Boolean expression
- `Logical Operator` can be one of those shown in **Table 15-2**

Table 15-2 Logical Operators in PHP

Logical Operator	Description
<code>&&</code>	AND (also known as logical conjunction)
<code> </code>	OR (also known as logical disjunction)
<code>!</code>	NOT (also known as negation or logical complement)

The truth table for all three operators is shown here.

Boolean Expression1 (BE1)	Boolean Expression2 (BE2)	BE1 && BE2	BE1 BE2	!(BE1)
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Are you still confused? You shouldn't be! It is quite simple!

- The result of the logical operator AND (`&&`) is `true` when both BE1 **and** BE2 are `true`.
- The result of the logical operator OR (`||`) is `true` when either BE1 **or** BE2 is `true` (at least one).
- The logical operator NOT (`!`) just reverses the result of a Boolean expression. In this table, when BE1 is `true` the result is `false` and vice versa.

Notice: In flowcharts, this book uses the commonly accepted AND, OR, and NOT operators!

Next are some examples of complex Boolean expressions. The parentheses are not really necessary. They are used just for increased readability.

- `($x == 5) && ($x > $y)`. This can be read as “test if `$x` is equal to 5 and greater than `$y`”
- `($x > $y) || ($x == 3)`. This can be read as “test if `$x` is greater than `$y` or equal to 3”
- `!($x < $y)`. This can be read as “test if `$x` is **not** less than `$y`” or, in other words, “test if `$x` is greater than or equal to `$y`”

Notice: In the last Boolean expression, please note the exclamation mark in the front.

15.5 What is the Order of Precedence of Logical Operators?

A very complex Boolean expression may use several logical operators like the one shown here

```
($x > $y) || ($x == 5) && ($x <= $z) || !($z == 1)
```

So, a reasonable question is “which logical operation is performed first?”

Logical operators follow the same precedence rules that apply to the majority of programming languages. The order of precedence is: logical complements (!) are performed first, logical conjunctions (&&) are performed next, and logical disjunctions (||) are performed at the end.

Higher Precedence	Logical Operator
	!
	&&
Lower Precedence	

Notice: You can always use parentheses to change the default precedence.

15.6 What is the Order of Precedence of Arithmetic, Comparison, and Logical Operators?

In many cases, an expression may contain different type of operators, such as the one shown here.

```
$a * $b + 2 > 21 || !($c == $b / 2) && $c > 13
```

In such cases, arithmetic operations are performed first, comparison operations are performed next, and logical operations are performed at the end, as shown in the following table

Higher Precedence  Lower Precedence	Arithmetic Operators	* / %
		+ -
	Comparison Operators	< <= > >= == !=
	Logical Operators	!
		&&

Exercise 15.6-1 *Filling in the Truth Table*

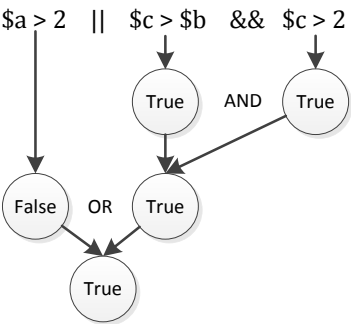
Fill in the following table with the words “True” or “False” according to the values of the variables \$a, \$b and \$c.

\$a	\$b	\$c	<code>\$a > 2 \$c > \$b && \$c > 2</code>	<code>!(\$a > 2 \$c > \$b && \$c > 2)</code>
1	-5	7		
10	10	3		
-4	-2	-9		

Solution

To calculate the result of complex Boolean expressions you can use the following graphical method.

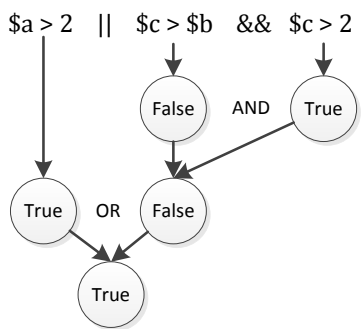
For \$a = 1, \$b = -5, \$c = 7,



the final result is true.

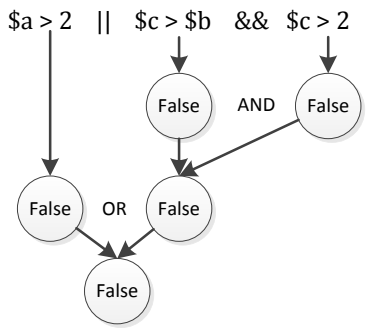
Remember! The AND operation (&&) has a higher precedence and is performed before the OR operation (||).

For \$a = 10, \$b = 10, \$c = 3,



the final result is true.

For \$a = -4, \$b = -2, \$c = -9,



the final result is false.

The values in the table’s fifth column can be calculated very easily because the Boolean expression in its column heading is almost identical to the one in the fourth column. The only difference is the NOT operator (!) in front of the expression. So, the values in the fifth column can be calculated by simply negating the results in the fourth column!

The final truth table is shown here.

\$a	\$b	\$c	\$a > 2 \$c > \$b && \$c > 2	!(\$a > 2 \$c > \$b && \$c > 2)
1	-5	7	True	False
10	10	3	True	False
-4	-2	-9	False	True

Exercise 15.6-2

Calculating the Results of Complex Boolean Expressions

Calculate the results of the following complex Boolean expressions when variables \$a, \$b, \$c, and \$d contain the values 5, 2, 7, and -3 respectively.

- i. (3 * \$a + \$b / 47 - \$c * \$b / \$a > 23) && (\$b != 2)
- ii. (\$a * \$b - \$c / 2 +21 * \$c / 3) || (\$a >= 5)

Solution

Don't be scared! The results can be found very easily. All you need is to recall what applies to AND (&&) and OR (||) operators.

- i. The result of an AND (&&) operator is `true` when both Boolean expressions are `true`. If you take a closer look, the result of the Boolean expression on the right (`$b != 2`) is `false`. So, you don't have to waste your time calculating the result of the Boolean expression on the left. The final result is definitely `false`.
- ii. The result of an OR (||) operator is `true` when at least one Boolean expression is `true`. If you take a closer look, the result of the Boolean expression on the right (`$a >= 5`) is actually `true`. So, don't bother calculating the result of the Boolean expression on the left. The final result is definitely `true`.

Exercise 15.6-3 Converting English Sentences to Boolean Expressions

A head teacher asks the students to raise their hands according to their age. He wants to find the students who are

- i. *between the ages of 9 and 12.*
- ii. *under the age of 8 and over the age of 11.*
- iii. *8, 10, and 12 years old.*
- iv. *between the ages of 6 and 8, and between the ages of 10 and 12.*
- v. *neither 10 nor 12 years old.*

Solution

To compose the required Boolean expressions, a variable `$age` is used.

- i. The first sentence can be graphically represented as shown here.



Be careful though! It is valid to write $9 \leq \text{age} \leq 12$ in mathematics, but in a computer language this is not possible. You must split the expression into two parts.

`$age >= 9 && $age <= 12`

For your confirmation, you can test this Boolean expression for several values inside and outside of the region of interest. For example, the result of the expression is `false` for the age values 7, 8, 13, and 17. On the contrary, for the age values 9, 10, 11, and 12, the result is `true`.

- ii. The second sentence can be graphically represented as shown here.



Notice: Please note the absence of the two circles. This means that the values 8 and 11 are not included within the two regions of interest.

Be careful with the sentence “Under the age of 8 and over the age of 11”. It’s a trap! Don’t make the mistake of writing

```
$age < 8 && $age > 11
```

There is no person on the planet Earth that can be under the age of 8 **and** over the age of 11 concurrently!

The trap is in the word “**and**”. Try to rephrase the sentence and make it “Children! Please raise your hand if you are under the age of 8 **or** over the age of 11.” Now it’s better and the correct Boolean expression becomes

```
$age < 8 || $age > 11
```

For your confirmation, you can test this expression for several values inside and outside of the regions of interest. For example, the result of the expression is *false* for the age values 8, 9, 10 and 11. On the contrary, for the age values 6, 7, 12, and 15, the result is *true*.

- iii. Oops! Another trap with the “**and**” word again! Obviously, the next Boolean expression is wrong.

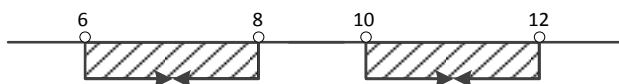
```
$age == 8 && $age == 10 && $age == 12
```

As before, there isn’t any student who is 8 **and** 10 **and** 12 years old concurrently! Once again, the correct Boolean expression must use the OR (||) operator.

```
$age == 8 || $age == 10 || $age == 12
```

For your confirmation, you can test this expression for several values inside and outside of the regions of interest. For example, the result of the expression is *false* for the age values 7, 9, 11, and 13. For the age values 8, 10, and 12, the result is *true*.

- iv. This sentence can be graphically represented as shown here.



and the Boolean expression is

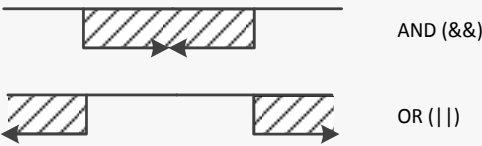
```
$age >= 6 && $age <= 8 || $age >= 10 && $age <= 12
```

For your confirmation, the result of the expression is *false* for the age values 5, 9, 13, and 16. For the age values 6, 7, 8, 10, 11, and 12, the result is *true*.

v. The Boolean expression for last sentence can be written as

```
$age != 10 && $age != 12
```

Remember! When the arrows of the region of interest are pointing toward each other, a logical operator AND (&&) must be used; otherwise, a logical operator OR (||) must be used.



15.7 How to Negate Boolean Expressions

Negation is the process of reversing the meaning of a Boolean expression. There are two approaches used to negate a Boolean expression.

First Approach

The first approach is the easiest one. Just use a NOT operator (!) in front of the original Boolean expression and your negated Boolean expression is ready! For example,

if the original Boolean expression is `$x > 5 && $y == 3`,
the negated Boolean expression becomes `!($x > 5 && $y == 3)`.

Notice: Please note that the entire expression must be enclosed in parentheses. It would be completely incorrect if you had written the expression without parentheses, as `!$x > 5 && $y == 3`. In this case the NOT operator (!) would negate only the first Boolean expression, `$x > 5`.

Second Approach

The second approach is a little bit more complex but not difficult to learn. All you must do is negate every operator according to the following table.

Original Operator	Negated Operator
==	!=
!=	==
>	<=
<	>=
<=	>
>=	<
&&	

	&&
!	!

Notice: Please note that the NOT operator (!) remains intact.

For example,
if the original Boolean expression is `$x > 5 && $y == 3`,
the negated Boolean expression becomes `$x <= 5 || $y != 3`.

Exercise 15.7-1 *Negating Boolean Expressions*

Negate the following Boolean expressions using both approaches.

- i. `$b != 4`

ii. `$a * 3 + 2 > 0`

iii. `!($a == 5 && $b >= 7)`
- iv. `$a == true`

v. `$b > 7 && !($x > 4)`

vi. `$a == 4 || $b != 2`

Solution

First Approach

- i. `!($b != 4)`

ii. `!($a * 3 + 2 > 0)`

iii. `!(($a == 5 && $b >= 7))`, or the equivalent `$a == 5 && $b >= 7`

Notice: Two negations result in an affirmative. That is, two NOT (!) operators in a row negate each other.

- iv. `!($a == true)`

v. `!($b > 7 && !($x > 4))`

vi. `!($a == 4 || $b != 2)`

Second Approach

- i. `$b == 4`

ii. `$a * 3 + 2 <= 0`

Notice: Please note that arithmetic operators are **not** negated. Don't you ever dare replace the plus (+) with a minus (-) operator!

- iii. `!($a != 5 || $b < 7)`

Notice: Please note that NOT operator (!) remains intact.

- iv. `$a != true`

v. `$b <= 7 || !($x <= 4)`

vi. `$a != 4 && $b == 2`

15.8 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A Boolean expression is an expression that always results in one of two values.
2. A Boolean expression includes at least one logical operator.
3. The expression `$x = 5` tests if the variable `$x` is equal to 5.
4. The statement

```
$a = $b > $c;
```

is not a valid PHP statement.

5. The Boolean expression `$b < 5` tests if the variable `$b` is 5 or less.
6. The AND (`&&`) operator is also known as a logical disjunction operator.
7. The NOT (`!`) operator is also known as a logical complement operator.
8. The result of a logical conjunction of two Boolean expressions equals the result of the logical disjunction of them, given that both Boolean expressions are true.
9. The result of a logical disjunction of two Boolean expressions is definitely true, given that the Boolean expressions have different values.
10. The expression `$c == 3 && $d > 7` is considered a complex Boolean expression.
11. The result of the logical operator OR (`||`) is true when both operands (Boolean expressions) are true.
12. The result of the Boolean expression `!($x == 5)` is true when the variable `$x` contains any value except 5.
13. The NOT (`!`) operator has the highest precedence among logical operators.
14. The OR (`||`) operator has the lowest precedence among logical operators.
15. In the Boolean expression `$x > $y || $x == 5 && $x <= $z`, the AND (`&&`) operation is performed before the OR (`||`) operation.
16. In the Boolean expression `$a * $b + $c > 21 || $c == $b / 2`, the program first tests if `$c` is greater than 21.
17. When a teacher wants to find the students who are under the age of 8 and over the age of 11, the corresponding Boolean expression is `$age < 8 && $age > 11`.
18. The Boolean expression `$x < 0 && $x > 100` is, for any value of `$x`, always false.
19. The Boolean expression `$x > 0 || $x < 100` is, for any value of `$x`, always true.
20. The Boolean expression `$x > 5` is equivalent to `!($x < 5)`.
21. The Boolean expression `!($x > 5 && $y == 5)` is not equivalent to `!($x > 5) && $y == 5`.

22. In William Shakespeare's *Hamlet* (Act 3, Scene 1), the main character says "To be, or not to be: that is the question:.... " If you write this down as a Boolean expression $2 * \$B \ || \ !(2 * \$B)$, the result of the "Shakespearean" expression is always `true` for the following code fragment.

```
$B = intval($That_is_the_question) % 2 / 2;  
$result = 2 * $B || !(2 * $B);
```

where variable `$That_is_the_question` contains numeric values.

23. The Boolean expression `!(($x > 5))` is equivalent to `$x > 5`.

15.9 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

- Which of the following is **not** a comparison operator?
 - `>=`
 - `=`
 - `<`
 - All of the above are comparison operators.
- Which of the following is not a logical operator?
 - `&`
 - `!`
 - All of the above are logical operators.
 - None of the above is a logical operator.
- If variable `$x` contains a value of 5, what value does the statement `$y = $x % 2 == 1` assign to variable `$y`?
 - `true`
 - `false`
 - none of the above
- If variable `$x` contains a value of 5, what value does the statement `$y = $x % 2 == 0 || intval($x / 2) == 2` assign to variable `$y`?
 - `true`
 - `false`
 - none of the above

¹ William Shakespeare (1564–1616) was an English poet, playwright, and actor. He is often referred to as England's national poet. He wrote about 40 plays and several long narrative poems. His works are counted among the best representations of world literature. His plays have been translated into every major living language and are still performed today.

5. The temperature in a laboratory room should be between 50 and 80 degrees Fahrenheit. Which of the following Boolean expressions tests for this condition?
- a. `$t >= 50 || $t <= 80`

b. `$t > 50 && $t < 80`

c. `$t >= 50 && $t <= 80`

d. `$t > 50 || $t < 80`
6. Which of the following is equivalent to the Boolean expression `$t == 3 || $t > 30` ?
- a. `$t == 3 && !($t <= 30)`

b. `$t == 3 && !($t < 30)`

c. `!($t != 3) || !($t < 30)`

d. `!($t != 3 && $t <= 30)`

15.10 Review Exercises

Complete the following exercises.

1. Match each element from the first column with one or more elements from the second column.

Operator	Sign
i. Logical operator	a. %
ii. Arithmetic operator	b. +=
iii. Comparison operator	c. &&
iv. Assignment operator (in general)	d. ==
	e.
	f. >=
	g. !
	h. =
	i. *=
	j. /

2. Fill in the following table with the words “True” or “False” according to the values of variables \$a, \$b, and \$c.

\$a	\$b	\$c	<code>\$a != 1</code>	<code>\$b > \$a</code>	<code>\$c / 2 > 2 * \$a</code>
3	-5	8			
1	10	20			
-4	-2	-9			

3. Fill in the following table with the words “True” or “False” according to the values of the Boolean expressions BE1 and BE2.

Boolean Expression1 (BE1)	Boolean Expression2 (BE2)	BE1 BE2	BE1 && BE2	!(BE2)
False	False			
False	True			
True	False			
True	True			

4. Fill in the following table with the words “True” or “False” according to the values of variables \$a, \$b, and \$c.

\$a	\$b	\$c	\$a > 3 \$c > \$b && \$c > 1	\$a > 3 && \$c > \$b \$c > 1
4	-6	2		
-3	2	-4		
2	5	5		

5. For \$x = 4, \$y = -2 and \$flag = true, fill in the following table with the corresponding values.

Expression	Value
pow(\$x + \$y, 3)	
(\$x + \$y) / (pow(\$x, 2) - 14)	
(\$x - 1) == \$y + 5	
\$x > 2 && \$y == 1	
\$x == 1 \$y == -2 && !(\$flag == false)	
!(\$x >= 3) && (\$x % 2 > 1)	

6. Calculate the result of each the following complex Boolean expressions when variables \$a, \$b, \$c, and \$d contain the values 6, -3, 4, and 7 respectively.

- $(3 * \$a + \$b / 5 - \$c * \$b / \$a > 4) \&\& (\$b != -3)$
- $(\$a * \$b - \$c / 2 + 21 * \$c / 3 != 8) || (\$a >= 5)$

7. A head teacher asks the students to raise their hands according to their age. He wants to find the students who are:

- under the age of 12, but not those who are 8 years old.
- between the ages of 6 and 9, and also those who are 11 years old.
- over the age of 7, but not those who are 10 or 12 years old.
- 6, 9, and 11 years old.

- v. between the ages of 6 and 12, but not those who are 8 years old.
- vi. neither 7 nor 10 years old.

To compose the required Boolean expressions, use a variable `$age`.

8. Negate the following Boolean expressions without using the NOT (!) operator.

- i. `$x == 4 && $y != 3`
- ii. `$x + 4 <= 0`
- iii. `!($x > 5) || $y == 4`
- iv. `$x != false`
- v. `!($x >= 4 || $z > 4)`
- vi. `$x != 2 && $x >= -5`

9. As you already know, two negations result in an affirmative. Write the equivalent of the following Boolean expressions by negating them twice (applying both methods).

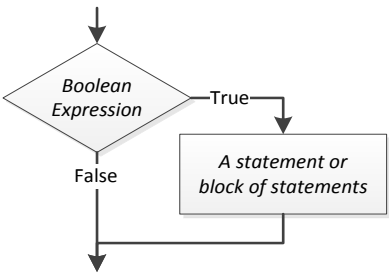
- i. `$x >= 4 && $y != 10`
- ii. `$x - 2 >= 9`
- iii. `!($x >= 2) || $y != 4`
- iv. `$x != false || $y == 3`
- v. `!($x >= 2 && $y >= 2)`
- vi. `$x != -2 && $x <= 2`

Chapter 16

The Single-Alternative Decision Structure

16.1 The Single-Alternative Decision Structure

This is the simplest decision control structure. It includes a statement or block of statements on the “true” path only.



If the Boolean expression evaluates to `true`, the statement, or block of statements, of the structure is executed; otherwise, the statements are skipped.

The general form of the PHP statement is

```
if (Boolean_Expression) {  
    A statement or block of statements  
}
```

Notice: In PHP, the Boolean expression must be enclosed in parentheses.

When only one single statement is used in the `if` statement, you can omit the braces `{ }`. Thus, the `if` statement becomes

```
if (Boolean_Expression)  
    One Single Statement;
```

or you can even write the statement on one single line, like this:

```
if (Boolean_Expression) One Single Statement;
```

The braces are required only when more than one statement is used in the `if` statement. Without the braces, PHP cannot tell whether a statement is part of the `if` statement or part of the statements that follow the `if` statement. Many programmers prefer to always use braces even if the `if` statement includes only one single statement. In both of the following examples the `echo $y` statement is **not** part of the `if` statement.

```
if ($x == $y)  
    $x++;  
    echo $y;
```

=

```
if ($x == $y) {  
    $x++;  
}  
echo $y;
```


Notice: A very common mistake that novice programmers make when writing PHP scripts is to confuse the assignment operator with the equal operator. They frequently make the mistake of writing `if ($x = $y)` when they actually want to say `if ($x == $y)`.

Exercise 16.1-1 Trace Tables and Single-Alternative Decision Structures

Design the corresponding flowchart and create a trace table to determine the values of the variables in each step of the next PHP script for two different executions.

The input values for the two executions are (i) 10, and (ii) 51.

file_16.1-1.php

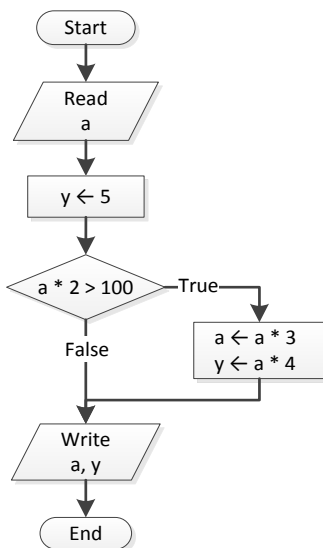
```
<?php
$a = trim(fgets(STDIN));

$y = 5;
if ($a * 2 > 100) {
    $a = $a * 3;
    $y = $a * 4;
}

echo $a, $y;
?>
```

Solution

The flowchart is shown here.



The trace tables for each input are shown here.

i. For the input value of 10, the trace table looks like this.

Step	Statement	Notes	\$a	\$y
1	<code>\$a = trim(fgets(STDIN))</code>	User enters the value 10	10	?
2	<code>\$y = 5</code>		10	5
3	<code>if (\$a * 2 > 100)</code>	This evaluates to false		
4	<code>echo \$a, \$y</code>	The values 10, 5 are displayed		

ii. For the input value of 51, the trace table looks like this.

Step	Statement	Notes	\$a	\$y
1	<code>\$a = trim(fgets(STDIN))</code>	User enters the value 51	51	?
2	<code>\$y = 5</code>		51	5
3	<code>if (\$a * 2 > 100)</code>	This evaluates to true		
4	<code>\$a = \$a * 3</code>		153	5
5	<code>\$y = \$a * 4</code>		153	612
6	<code>echo \$a, \$y</code>	The values 153, 612 are displayed		

Exercise 16.1-2 *The Absolute Value of a Number*

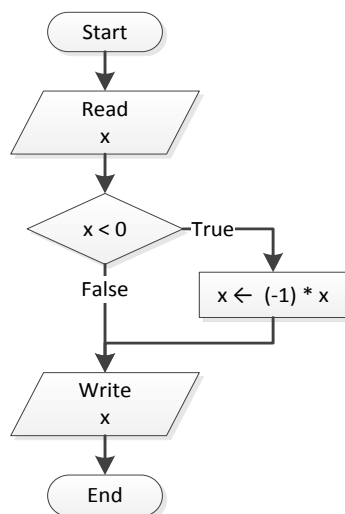
Design a flowchart and write the corresponding PHP script that lets the user enter a number and then displays its absolute value.

Solution

Actually, there are two approaches. The first approach uses a single-alternative decision structure, whereas the second one uses the built-in `abs()` function.

First Approach – Using a single-alternative decision structure

The approach is simple. If the user enters a negative value, for example -5, this value is changed and displayed as +5. A positive number or zero, however, should remain as is. The solution is shown in the flowchart that follows.



The corresponding PHP script is as follows:

file_16.1-2a.php

```
<?php
    $x = trim(fgets(STDIN));

    if ($x < 0) {
        $x = (-1) * $x;
    }

    echo $x;
?>
```

Second Approach – Using the `abs()` function

In this case, you need just two lines of code without any decision control structure!

file_16.1-2b.php

```
<?php
    $x = trim(fgets(STDIN));
    echo abs($x);
?>
```

16.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The single-alternative decision structure is used when a sequence of statements must be executed.
2. You use a single-alternative decision structure to allow other programmers to more easily understand your script.

3. It is possible that none of the statements enclosed in a single-alternative decision structure will be executed.
4. In a flowchart, the Decision symbol represents the beginning and the end of an algorithm.
5. The code fragment

```
define("if", 5);  
$x = if + 5;
```

is syntactically correct.

6. The single-alternative decision structure uses the reserved keyword `else`.
7. The following code fragment satisfies the property of definiteness.

```
if ($b != 3) {  
    $x = $a / ($b - 3);  
}
```

8. The following PHP script satisfies the property of definiteness.

```
<?php  
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
  
if ($b != 3) {  
    $x = $a / ($b - 3);  
}  
echo $x;  
?>
```

16.3 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The single-alternative decision structure is used when
 - a. statements are executed one after another.
 - b. a decision must be made before executing some statements.
 - c. none of the above
 - d. all of the above
2. The single-alternative decision structure includes a statement or block of statements on
 - a. the false path only.
 - b. both paths.
 - c. the true path only.

3. In the following code fragment,

```
if ($x == 3)
    $x = 5;
    $y++;
```

the statement `$y++` is executed

- only when variable `$x` contains a value of 3.
 - only when variable `$x` contains a value of 5.
 - only when variable `$x` contains a value other than 3.
 - either way.
4. In the following code fragment,

```
if ($x % 2 == 0) $y++;
```

the statement `$y++` is executed when

- variable `$x` is exactly divisible by 2.
 - variable `$x` contains an even number.
 - variable `$x` does not contain an odd number.
 - all of the above
 - none of the above
5. In the following code fragment,

```
$x = 3 * $y;
if ($x > $y) $y++;
```

the statement `$y++` is

- always executed.
- never executed.
- executed either way.
- executed only when variable `$y` contains positive values.
- none of the above

16.4 Review Exercises

Complete the following exercises.

1. Identify the syntax errors in the following PHP script:

```
<?php
    $x = trim(fgets(STDIN));

    $y ← - 5;
    if ($x * $y / 2 > 20)
        $y--
        $x = - 4 * $x2;
}
```

```
echo $x $y;  
?>
```

2. Create a trace table to determine the values of the variables in each step of the following PHP script for two different executions. Then, design the corresponding flowchart.

The input values for the two executions are (i) 10, and (ii) -10.

```
<?php  
$x = trim(fgets(STDIN));  
  
$y = - 5;  
if ($x * $y / 2 > 20) {  
    $y--;  
    $x -= 4;  
}  
if ($x > 0) {  
    $y += 30;  
    $x = pow($x, 2);  
}  
  
echo $x, $y;  
?>
```

3. Create a trace table to determine the values of the variables in each step of the following PHP script for two different executions. Then, design the corresponding flowchart.

The input values for the two executions are (i) -11, and (ii) 11.

```
<?php  
$x = trim(fgets(STDIN));  
  
$y = 8;  
if (abs($x) > 10) {  
    $y += $x;  
    $x--;  
}  
if (abs($x) > 10) {  
    $y *= 3;  
}  
  
echo $x, $y;  
?>
```

4. Create a trace table to determine the values of the variables in each step of the following PHP script for two different executions. Then, design the corresponding flowchart.

The input values for the two executions are (i) 1, 2, 3; and (ii) 4, 2, 1.

```
<?php
    $x = trim(fgets(STDIN));
    $y = trim(fgets(STDIN));
    $z = trim(fgets(STDIN));

    if ($x + $y > $z)
        $x = $y + $z;
    if ($x > $y + $z)
        $y = $x + $z;
    if ($x > $y - $z)
        $z = $x - $z % 2;

    echo $x, $y, $z;
?>
```

5. Write a PHP script that prompts the user to enter a number, and then displays the message “Positive” when the given number is positive.
6. Write a PHP script that prompts the user to enter two numbers, and then displays the message “Positive” when both given numbers are positives.
7. Write a PHP script that prompts the user to enter a value and then displays the message “Numeric” when the given value is a numeric value.
Hint: Use the `is_numeric()` function.
8. Write a PHP script that prompts the user to enter a string, and then displays the message “Uppercase” when the given string contains only uppercase characters.
Hint: Use the `strtoupper()` function
9. Write a PHP script that prompts the user to enter a string, and then displays the message “Many characters” when the given string contains more than 20 characters.
Hint: Use the `strlen()` function
10. Write a PHP script that prompts the user to enter four numbers and, if one of them is negative, it displays the message “Among the given numbers, there is a negative one!”
11. Write a PHP script that prompts the user to enter two numbers. If the first number given is greater than the second one, the script should swap their values. In the end, the script should display the numbers, always in ascending order.

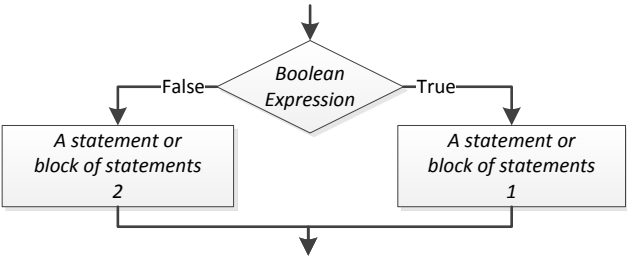
12. Write a PHP script that prompts the user to enter three temperature values measured at three different points in New York, and then displays the message “Heat Wave” if the average value is greater than 60 degrees Fahrenheit.

Chapter 17

The Dual-Alternative Decision Structure

17.1 The Dual-Alternative Decision Structure

This type of decision control structure includes a statement or block of statements on both paths.



If the Boolean expression evaluates to `true`, the statement or block of statements 1 is executed; otherwise, the statement or block of statements 2 is executed.

The general form of the PHP statement is

```
if (Boolean_Expression) {  
    A statement or block of statements 1  
}  
else {  
    A statement or block of statements 2  
}
```

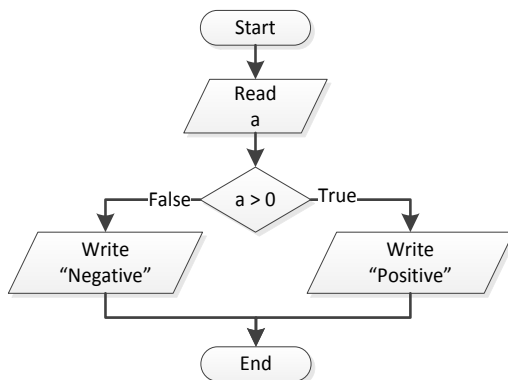
Once again, single statements can be written without being enclosed inside braces {}, and the `if-else` statement becomes

```
if (Boolean_Expression)  
    One Single Statement 1;  
else  
    One Single Statement 2;
```

Exercise 17.1-1 Finding the Output Message

For the following flowchart, determine the output message for three different executions.

The input values for the three executions are (i) 3, (ii) -3, and (iii) 0.



Solution

- When the user enters the value 3, the Boolean expression evaluates to `true`. The flow of execution follows the right path and the message “Positive” is displayed.
- When the user enters the value -3, the Boolean expression evaluates to `false`. The flow of execution follows the left path and the message “Negative” is displayed.
- When the user enters the value 0, the Boolean expression evaluates to `false`. Inevitably, the flow of execution follows the left path again and the message “Negative” is displayed!

Notice: Obviously, something is not quite right with this flowchart! As you already know, zero is not a negative value! In fact, it is not a positive value either. Later in this book, when you learn all about nested decision control structures, you will learn how to display a third message that will say “The number entered is zero”.

Remember! A Decision symbol has one entrance and two exit paths! You cannot have a third exit!

Exercise 17.1-2 Trace Tables and Dual-Alternative Decision Structures

Create a trace table to determine the values of the variables in each step of the next PHP script for two different executions.

The input values for the two executions are (i) 5, and (ii) 10.

file_17.1-2.php

```
<?php
```

```
$a = trim(fgets(STDIN));
```

```
$z = $a * 10;
```

```
$w = ($z - 4) * ($a - 3) / 7 + 36;
```

```
if ($z >= $w && $a < $z) {
    $y = 2 * $a;
}
else {
    $y = 4 * $a;
}

echo $y;

?>
```

Solution

i. For the input value of 5, the trace table looks like this.

Step	Statement	Notes	\$a	\$z	\$w	\$y
1	\$a = trim(fgets(STDIN))	User enters the value 5	5	?	?	?
2	\$z = \$a * 10		5	50	?	?
3	\$w = (\$z - 4) * (\$a - 3) / 7 + 36		5	50	49.142	?
4	if (\$z >= \$w && \$a < \$z)	This evaluates to true				
5	\$y = 2 * \$a		5	50	49.142	10
6	echo \$y	The value 10 is displayed				

ii. For the input value of 10, the trace table looks like this.

Step	Statement	Notes	\$a	\$z	\$w	\$y
1	\$a = trim(fgets(STDIN))	User enters the value 10	10	?	?	?
2	\$z = \$a * 10		10	100	?	?
3	\$w = (\$z - 4) * (\$a - 3) / 7 + 36		10	100	132	?
4	if (\$z >= \$w && \$a < \$z)	This evaluates to false				
5	\$y = 4 * \$a		10	100	132	40
6	echo \$y	The value 40 is displayed				

Exercise 17.1-3 *Who is the Greatest?*

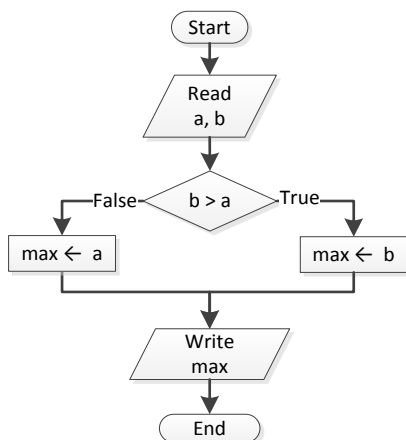
Design a flowchart and write the corresponding PHP script that lets the user enter two numbers A and B and then determines and displays the greater of the two numbers.

Solution

This exercise can be solved using either the single- or dual-alternative decision structure. So, let's use them both!

First Approach – Using a dual-alternative decision structure

This approach tests if the value of number B is greater than that of number A. If so, number B is the greatest; otherwise, number A is the greatest.



and the PHP script is shown here.

file_17.1-3a.php

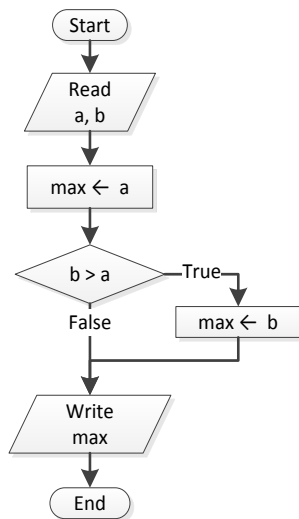
```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

if ($b > $a) {
    $max = $b;
}
else {
    $max = $a;
}

echo "Greatest value: ", $max;
?>
```

Second Approach – Using a single-alternative decision structure

This approach initially assumes that number A is probably the greatest value (this is why it assigns the value of variable \$a to variable \$max). However, if it turns out that number B is actually greater than number A, then the greatest value is updated, that is, variable \$max is assigned a new value, the value of variable \$b. Thus, whatever happens, in the end, variable \$max always contains the greatest value!



Notice: Please note that this exercise is trying to determine the greatest value and not which variable this value is actually assigned to (to variable A or to variable B).

The PHP script is shown here.

file_17.1-3b.php

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$max = $a;
if ($b > $a) {
    $max = $b;
}

echo "Greatest value: ", $max;
?>
```

Exercise 17.1-4 *Finding Odd and Even Numbers*

Design a flowchart and write the corresponding PHP script that prompts the user to enter an integer, and then displays a message indicating whether this number is odd or even.

Solution

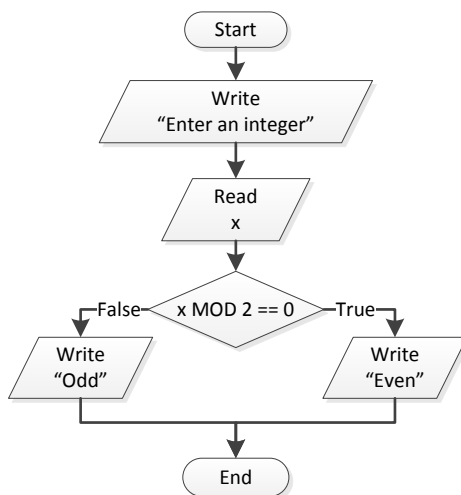
In this exercise, you need to find a way to determine whether a number is odd or even. You need to find a common attribute between all even numbers, or between all odd numbers. And actually there is one! All even numbers are exactly divisible by 2. So, when the result of the operation $x \% 2$ equals 0, x is even; otherwise, x is odd.

Next you can find various odd and even numbers:

- Odd numbers: 1, 3, 5, 7, 9, 11, ...
- Even numbers: 0, 2, 4, 6, 8, 10, 12, ...

Notice: Please note that zero is considered an even number.

The flowchart is shown here.



and the PHP script is shown here.

file_17.1-4.php

```
<?php
echo "Enter an integer: ";
$x = trim(fgets(STDIN));

if ($x % 2 == 0) {
    echo "Even";
}
else {
```

```
    echo "Odd";  
}  
?>
```

Exercise 17.1-5 Weekly Wages

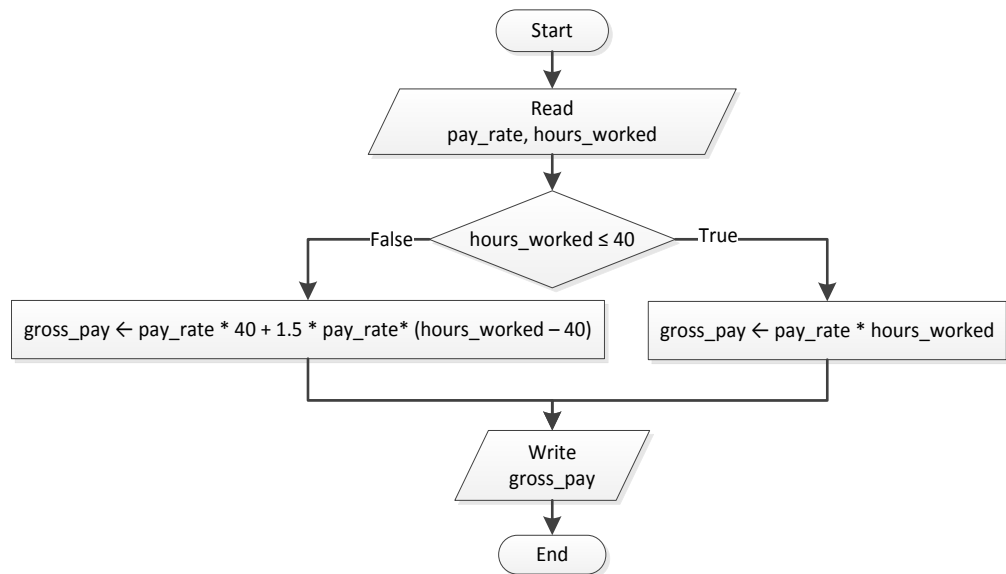
Gross pay depends on the pay rate and the total number of hours worked per week. However, if someone works more than 40 hours, he or she gets paid time-and-a-half for all hours worked over 40. Design a flowchart and write the corresponding PHP script that lets the user enter a pay rate and the hours worked and then calculates and displays the gross pay.

Solution

This exercise can be solved using the dual-alternative decision structure. When the hours worked are over 40, the gross pay is calculated as follows:

$$\text{gross pay} = (\text{pay rate}) \times 40 + 1.5 \times (\text{pay rate}) \times (\text{all hours worked over } 40)$$

The flowchart that solves this problem is shown here.



and the PHP script is shown here.

file_17.1-5.php

```
<?php  
$pay_rate = trim(fgets(STDIN));  
$hours_worked = trim(fgets(STDIN));  
  
if ($hours_worked <= 40) {
```



```
$gross_pay = $pay_rate * $hours_worked;
}
else {
    $gross_pay = $pay_rate * 40 + 1.5 * $pay_rate * ($hours_worked - 40);
}

echo "Gross Pay: ", $gross_pay;

?>
```

17.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. It is possible that none of the statements enclosed in a dual-alternative decision structure will be executed.
2. The dual-alternative decision structure must include at least two statements.
3. The dual-alternative decision structure uses the reserved keyword `else`.
4. The code fragment

```
define("else", 5);
if (else == 5) echo "hello";
```

is syntactically correct.

5. In a dual-alternative decision structure, the evaluated Boolean expression can return more than two values.
6. The following code fragment satisfies the property of effectiveness.

```
$x = trim(fgets(STDIN));
$y = trim(fgets(STDIN));
$z = trim(fgets(STDIN));

if ($x > $y && $x > $z) {
    echo "Value ", $x, " is the greatest one";
}
else {
    echo "Value ", $y, " is the greatest one";
}
```

17.3 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The dual-alternative decision structure includes a statement or block of statements on
 - a. the false path only.
 - b. both paths.

c. the true path only.

2. In the following code fragment,

```
if ($x % 2 == 0) {  
    $x = 0;  
}  
else {  
    $y++;  
}
```

the statement `$y++` is executed when

- a. variable `$x` is exactly divisible by 2.
- b. variable `$x` contains an even number.
- c. variable `$x` contains an odd number.
- d. none of the above

3. In the following code fragment,

```
if ($x == 3)  
    $x = 5;  
else  
    $x = 7;  
    $y++;
```

the statement `$y++` is executed

- a. when variable `$x` contains a value of 3.
- b. when variable `$x` contains a value other than 3.
- c. both of the above

17.4 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the next PHP script for two different executions. Then, design the corresponding flowchart.

The input values for the two executions are (i) 3, and (ii) 0.5.

```
<?php  
$a = trim(fgets(STDIN));  
$z = $a * 3 - 2;  
if ($z >= 1) {  
    $y = 6 * $a;  
}  
else {  
    $z++;  
    $y = 6 * $a + $z;
```

```

    }
    echo $z, $y;
?>

```

2. Create a trace table to determine the values of the variables in each step of the next PHP script. Then, design the corresponding flowchart.

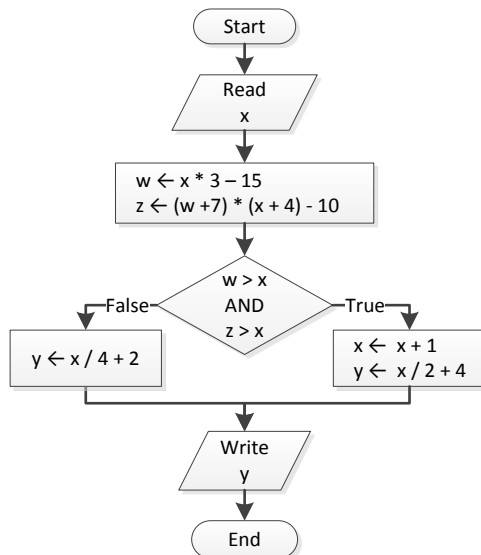
```

<?php
    $x = 3;
    $y = pow($x, 3) + 9;
    $z = 2 * $x + $y - 4;
    if ($x > $y) {
        $y = $z % $x;
        $z = sqrt($x);
    }
    else {
        $x = $z % $y;
        $z = sqrt($y);
    }
    echo $x, $y, $z;
?>

```

3. Write the PHP script that corresponds to the following flowchart and then create a trace table to determine the values of the variables in each step for two different executions.

The input values for the two executions are (i) 10, and (ii) 2,



4. Write a PHP script that lets the user enter an integer, and then displays a message indicating whether or not the given number is a multiple of 6.
5. Write a PHP script that lets the user enter an integer, and then displays one of two possible messages. One message indicates if the given number is a multiple of 6 or a multiple of 7; the other message indicates if the given number is neither a multiple of 6 nor a multiple of 7.
6. Write a PHP script that lets the user enter an integer, and then displays a message indicating whether or not the given number is a multiple of 4. Moreover, the PHP script should display the structure of the given integer, including the given integer, the quotient, and any remainder. For example, if the given integer is 14, the message “ $14 = 3 \times 4 + 2$ ” should be displayed.
7. Write a PHP script that lets the user enter an integer, and then displays a message indicating whether or not the given integer is a four-digit integer.
Hint: Four-digit integers are between 1000 and 9999.
8. Design a flowchart and write the corresponding PHP script that lets the user enter two values, and then determines and displays the smaller of the two values. Assume that the user enters two different values.
9. Write a PHP script that lets the user enter three numbers, and then displays a message indicating whether or not the given numbers can be lengths of the three sides of a triangle.
Hint: In any triangle, the length of each side is less than the sum of the lengths of the other two sides.
10. Write a PHP script that lets the user enter three numbers, and then displays a message indicating whether or not the given numbers can be lengths of the three sides of a right triangle (or right-angled triangle). You can use lengths of 3, 4 and 5 to test your script.
Hint: Use the Pythagorean theorem.
11. Athletes in the long jump at the Olympic Games in Athens in 2004 participated in three different qualifying jumps. Assume that in order to qualify, an athlete had to achieve an average jump distance of 8 meters. Write a PHP script that prompts the user to enter the three performances, and then displays the messages “Qualified” or “Disqualified” depending on whether the average value is greater or less than 8 meters.
12. Gross pay depends on the pay rate and the total number of hours worked per week. However, if someone works more than 40 hours, he or she gets paid double for all hours worked over 40. Design a flowchart and write the corresponding PHP script that lets the user enter the pay rate and hours worked and then calculates and displays net pay. Net pay is the amount of pay that is actually paid to the employee after any deductions. Deductions include taxes, health insurance, retirement plans, on so on. Assume a total deduction of 30%.

13. Regular servicing will keep your vehicle more reliable, reducing the chance of breakdowns, inconvenience and unnecessary expenses. In general, there are two types of service you need to perform:
- a. a minor service every 6000 miles
 - b. a major service every 12000 miles

Write a PHP script that prompts the user to enter the miles traveled, and then calculates and displays how many miles are left until the next service, as well as the type of the next service.

14. Two cars start from rest and move with a constant acceleration along a straight horizontal road for a given time. Write a PHP script that prompts the user to enter the time the two cars traveled (same for both cars) and the acceleration for each one of them, and then calculates and displays the distance between them as well as a message “Car A is first” or “Car B is first” depending on which car is leading the race. The required formula is

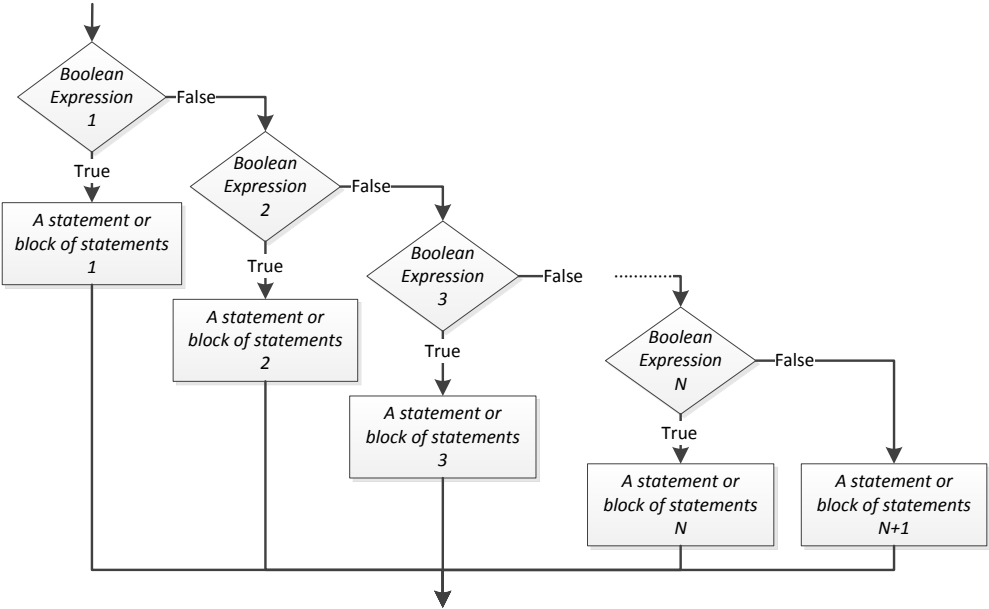
$$S = u_o + \frac{1}{2}at^2$$

Chapter 18

The Multiple-Alternative Decision Structure

18.1 The Multiple-Alternative Decision Structure

The multiple-alternative decision structure is used to expand the number of alternatives, as shown in the following flowchart fragment.



When a multiple-alternative decision structure is executed, *Boolean Expression 1* is evaluated. If *Boolean Expression 1* evaluates to *true*, the corresponding statement or block of statements that immediately follows it is executed and the rest of the structure is skipped. In other words, the flow of execution continues to any remaining statements that may exist **after** the multiple-alternative decision structure. However, if *Boolean Expression 1* evaluates to *false*, the flow of execution evaluates *Boolean Expression 2*. If it evaluates to *true*, the corresponding statement or block of statements that immediately follows it is executed and the rest of the structure is skipped. This process continues until one Boolean expression evaluates to *true* or until no more Boolean expressions are left.

The last statement or block of statements *N+1* is executed when none of the previous Boolean expressions has evaluated to *true*. Moreover, this last statement or block of statements *N+1* is optional and can be omitted. It depends on the algorithm you are trying to solve.

The general form of the PHP statement is

```
if (Boolean_Expression 1) {  
    A statement or block of statements 1  
}  
  
elseif (Boolean_Expression 2) {  
    A statement or block of statements 2
```

```
}  
elseif (Boolean_Expression 3) {  
    A statement or block of statements 3  
}  
.  
.  
.  
elseif (Boolean_Expression N) {  
    A statement or block of statements N  
}  
else {  
    A statement or block of statements N+1  
}
```

As you already know, single statements can be written without being enclosed in braces {}. Thus, the if-elseif statement would become as shown here.

```
if (Boolean_Expression 1)  
    One Single Statement 1  
elseif (Boolean_Expression 2)  
    One Single Statement 2  
elseif (Boolean_Expression 3)  
    One Single Statement 3  
.  
.  
.  
elseif (Boolean_Expression N)  
    One Single Statement N  
else  
    One Single Statement N+1
```

Exercise 18.1-1 *Trace Tables and Multiple-Alternative Decision Structures*

Create a trace table to determine the values of the variables in each step for three different executions of the next PHP script.

The input values for the three executions are (i) 5, 8; (ii) -13, 0; and (iii) 1, -1.

file_18.1-1.php

```
<?php  
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));
```



```
if ($a > 3)
    echo "Variable A greater than 3";
elseif ($a > 4 && $b <= 10) {
    echo "Variable A greater than 4 ";
    echo "and B less than or equal to 10";
}
elseif ($a * 2 == -26) {
    echo "Variable A times 2 is equal to -26";
    $b++;
}
elseif ($b == 1)
    echo "Variable B is equal to 1";
else
    echo "Display this when none of the above evaluates to true";

echo "The end!";

?>
```

Notice: Please note that you can use braces { } only when necessary.

Solution

i. For the input values of 5 and 8, the trace table looks like this.

Step	Statement	Notes	\$a	\$b
1	\$a = trim(fgets(STDIN))	User enters the value 5	5	?
2	\$b = trim(fgets(STDIN))	User enters the value 8	5	8
3	if (\$a > 3)	This evaluates to true		
4	echo "Variable A greater ...	The message "Variable A greater than 3" is displayed		
5	echo "The end!"	The message "The end!" is displayed		

Notice: Please note that even though the second Boolean expression (\$a > 4 && \$b <= 10) could also have evaluated to true, it was not checked.

ii. For the input values of -13 and 0, the trace table looks like this.

Step	Statement	Notes	\$a	\$b
1	\$a = trim(fgets(STDIN))	User enters the value -13	-13	?
2	\$b = trim(fgets(STDIN))	User enters the value 0	-13	0
3	if (\$a > 3)	This evaluates to false		

4	elseif (\$a > 4 && \$b <= 10)	This evaluates to false		
5	elseif (\$a * 2 == -26)	This evaluates to true		
6	echo "Variable A times 2..."	The message "Variable A times 2 is equal to -26" is displayed		
7	\$b++		-13	1
8	echo "The end!"	Message "The end!" is displayed		

Notice: Please note that after step 7 the fourth Boolean expression (`$b == 1`) could also have evaluated to `true`, but it was not checked.

iii. For the input values of 1 and -1, the trace table looks like this.

Step	Statement	Notes	\$a	\$b
1	<code>\$a = trim(fgets(STDIN))</code>	User enters the value 1	1	?
2	<code>\$b = trim(fgets(STDIN))</code>	User enters the value -1	1	-1
3	<code>if (\$a > 3)</code>	This evaluates to false		
4	<code>elseif (\$a > 4 && \$b <= 10)</code>	This evaluates to false		
5	<code>elseif (\$a * 2 == -26)</code>	This evaluates to false		
6	<code>elseif (\$b == 1)</code>	This evaluates to false		
7	<code>echo "Display this ..."</code>	The message "Display this when none of the above evaluates to true" is displayed		
8	<code>echo "The end!"</code>	The message "The end!" is displayed		

Exercise 18.1-2 *Counting the Digits*

Write a PHP script that prompts the user to enter an integer between 0 and 999 and then counts its total number of digits. In the end, a message “You entered a N-digit number” is displayed, where N is the total number of digits.

Solution

The following PHP script assumes that the user enters a valid number between 0 and 999, and therefore does not include checking the validity of data input.

file_18.1-2.php

```
<?php
    echo "Enter a number (0 - 999): ";
    $x = trim(fgets(STDIN));

    if ($x <= 9) {
```

```
$count = 1;
}
elseif ($x <= 99) {
    $count = 2;
}
else {
    $count = 3;
}

echo "You entered a ", $count, "-digit number";
?>
```

18.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The multiple-alternative decision structure is used to expand the number of alternatives.
2. The multiple-alternative decision structure can have at most three alternatives.
3. In a multiple-alternative decision structure, once a Boolean expression evaluates to **true**, the next Boolean expression is evaluated.
4. In a multiple-alternative decision structure, the last statement or block of statements N+1 (appearing below the `else` PHP keyword) is always executed.
5. In a multiple-alternative decision structure, the last statement or block of statements N+1 (appearing below the `else` PHP keyword) is executed when at least one of the previous Boolean expressions has evaluated to **true**.
6. In a multiple-alternative decision structure, the last statement or block of statements N+1, and by extension the `else` PHP keyword, can be omitted.
7. In the following code fragment,

```
if ($a == 1)
    $x = $x + 5;
elseif ($a == 2)
    $x = $x - 2;
elseif ($a == 3)
    $x = $x - 9;
else
    $x = $x + 3;
$y++;
```

the statement `$y++` is executed only when variable `$a` contains a value other than 1, 2, or 3.

18.3 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step for four different executions of the next PHP script.

The input values for the four executions are (i) 5, (ii) 150, (iii) 250, and (iv) -1.

```
<?php
    $q = trim(fgets(STDIN));

    if ($q > 0 && $q <= 50) {
        $b = 1;
    }
    elseif ($q > 50 && $q <= 100) {
        $b = 2;
    }
    elseif ($q > 100 && $q <= 200) {
        $b = 3;
    }
    else {
        $b = 4;
    }
    echo $b;
?>
```

2. Create a trace table to determine the values of the variables in each step for three different executions of the next PHP script.

The input values for the three executions are (i) 5, (ii) 150, and (iii) -1.

```
<?php
    $amount = trim(fgets(STDIN));

    if ($amount < 20) {
        $discount = 0;
    }
    elseif ($amount >= 20 && $amount < 60) {
        $discount = 5;
    }
    elseif ($amount >= 60 && $amount < 100) {
        $discount = 10;
    }
    elseif ($amount >= 100) {
```

```
$discount = 15;
}
$payment = $amount - $amount * $discount / 100;

echo $discount, $payment;
?>
```

3. Design a flowchart and write the corresponding PHP script that lets the user enter an integer between -9999 and 9999, and then counts its total number of digits. In the end, a message “You entered a N-digit number” is displayed, where N is the total number of digits.
4. Write a PHP script that displays the following menu:
 1. Convert USD to Euro (EUR)
 2. Convert USD to British Pound Sterling (GBP)
 3. Convert USD to Japanese Yen (JPY)
 4. Convert USD to Canadian Dollar (CAD)

It then prompts the user to enter a choice (of 1, 2, 3, or 4) and an amount in US dollars and calculates and displays the required value. It is given that

- \$1 = 0.72 EUR (€)
- \$1 = 0.60 GBP (£)
- \$1 = ¥ 102.15 JPY
- \$1 = 1.10 CAD (\$)

5. Write a PHP script that prompts the user to enter the number of a month between 1 and 12, and then displays the corresponding season. It is given that
 - Winter includes months 12, 1, and 2
 - Spring includes months 3, 4, and 5
 - Summer includes months 6, 7, and 8
 - Fall (Autumn) includes months 9, 10, and 11
6. Write a PHP script that prompts the user to enter a number between 1.0 and 4.9, and then displays the number as English text. For example, for the number 2.3, it should display “Two point three”.
7. The most popular and commonly used grading system in the United States uses discrete evaluation in the form of letter grades. Design a flowchart and write the corresponding PHP script that prompts the user to enter a letter between A and F, and then displays the corresponding percentage according to the following table.

Grade	Percentage
A	90 – 100
B	80 – 89

C	70 – 79
D	60 – 69
E / F	0 – 59

Chapter 19

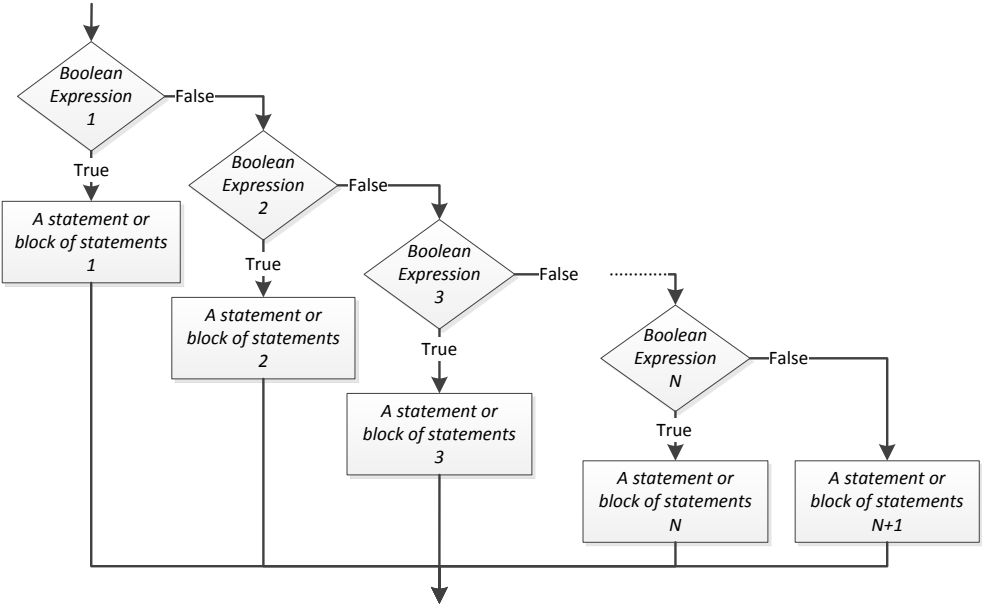
The Case Decision Structure

19.1 The Case Decision Structure

The case decision structure is a simplified version of the multiple-alternative decision structure. It helps you write code faster and increases readability, especially for algorithms that require complex combinations of decision structures.

The case decision structure is used to expand the number of alternatives in the same way as the multiple-alternative decision structure does. The big difference between them is that the evaluated variable or expression in a case decision structure must be the same in all cases.

The case decision structure and the multiple-alternative decision structure even share the same flowchart.



As in the multiple-alternative decision structure, when a case decision structure is executed, *Boolean Expression 1* is evaluated. If *Boolean Expression 1* evaluates to **true**, the corresponding statement or block of statements that immediately follows it is executed and the rest of the structure is skipped. In other words, the flow of execution continues to any remaining statements that may exist **after** the multiple-alternative decision structure. However, if *Boolean Expression 1* evaluates to **false**, the flow of execution evaluates *Boolean Expression 2*. If it evaluates to **true**, the corresponding statement or block of statements that immediately follows it is executed and the rest of the structure is skipped. This process continues until one Boolean expression evaluates to **true** or until no more Boolean expressions are left.

The last statement or block of statements *N+1* is executed when none of the previous Boolean expressions has evaluated to **true**. Moreover, this last statement or block of statements *N+1* is optional and can be omitted. It depends on the algorithm that you are trying to solve.

The general form of the PHP statement is

```
switch (a variable or an expression to evaluate) {  
    case value-1:  
        A statement or block of statements 1  
        break;  
    case value-2:  
        A statement or block of statements 2  
        break;  
    case value-3:  
        A statement or block of statements 3  
        break;  
    .  
    .  
    .  
    case value-N:  
        A statement or block of statements N  
        break;  
    default:  
        A statement or block of statements N+1  
}
```

Notice: The last statement or last block of statements $N+1$ is optional and can be omitted (you need to omit the keyword `default` as well).

Notice: In order to avoid undesirable results, please remember to always include the keyword `break` at the end of each case. If you omit one, two statements or blocks of statements are actually executed: the current one in which the keyword `break` is omitted, and the next one.

An example that uses the case decision structure is shown here.

file_19.1.php

```
<?php  
echo "What is your name? ";  
$name = trim(fgets(STDIN));  
  
switch ($name) {  
    case "John":  
        echo "You are my cousin!";  
        break;  
    case "Aphrodite":  
        echo "You are my sister!";  
        break;
```

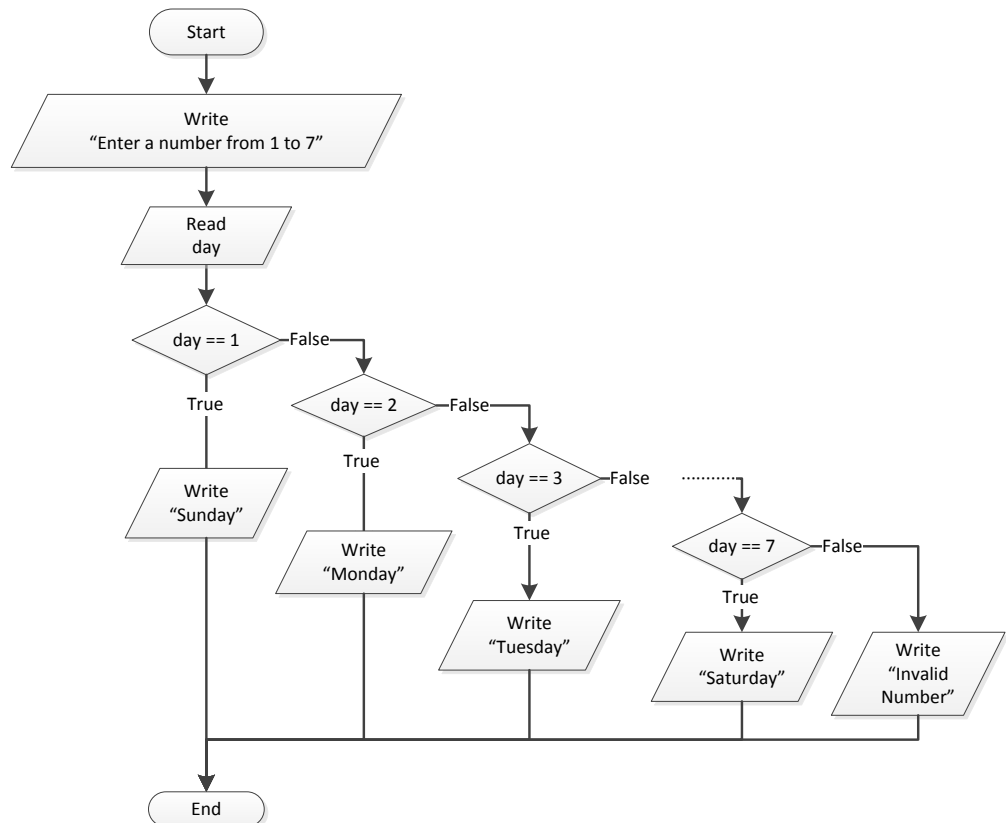
```
case "Loukia":  
    echo "You are my mom!";  
    break;  
default:  
    echo "Sorry, I don't know you.";  
}  
?>
```

Exercise 19.1-1 *The Days of the Week*

Design a flowchart and write the corresponding PHP script that prompts the user to enter a number between 1 and 7, and then displays the corresponding day (Sunday, Monday, and so on). If the value entered is invalid, an error message should be displayed.

Solution

The flowchart is shown here.



This PHP script can be written using either a case decision structure or a multiple-alternative decision structure. Let's try them both!

First Approach – Using a case decision structure

file_19.1-1a.php

```
<?php
echo "Enter a number between 1 and 7: ";
$day = trim(fgets(STDIN));

switch ($day) {
    case 1:
        echo "Sunday";
        break;
    case 2:
        echo "Monday";
        break;
    case 3:
        echo "Tuesday";
        break;
    case 4:
        echo "Wednesday";
        break;
    case 5:
        echo "Thursday";
        break;
    case 6:
        echo "Friday";
        break;
    case 7:
        echo "Saturday";
        break;
    default:
        echo "Invalid Number";
}
?>
```

Second Approach – Using a multiple-alternative decision structure

file_19.1-1b.php

```
<?php
echo "Enter a number between 1 and 7: ";
$day = trim(fgets(STDIN));
```

```
if ($day == 1) {  
    echo "Sunday";  
}  
elseif ($day == 2) {  
    echo "Monday";  
}  
elseif ($day == 3) {  
    echo "Tuesday";  
}  
elseif ($day == 4) {  
    echo "Wednesday";  
}  
elseif ($day == 5) {  
    echo "Thursday";  
}  
elseif ($day == 6) {  
    echo "Friday";  
}  
elseif ($day == 7) {  
    echo "Saturday";  
}  
else {  
    echo "Invalid Number";  
}  
?  
>
```

19.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The case decision structure is used to expand the number of alternatives.
2. The case decision structure can always be used instead of a multiple-alternative decision structure.
3. The case decision structure can have as many alternatives as the programmer wishes.
4. In a case decision structure, the last statement or block of statements N+1 (appearing below the default PHP keyword) is always executed.
5. In a case decision structure, the last statement or block of statements N+1 (appearing below the default PHP keyword) is executed when none of the previous cases has evaluated to true.
6. The last statement or block of statements N+1, as well as the default PHP keyword, cannot be omitted.

7. In the following PHP script,

```
switch ($a) {  
    case 1: $x = $x + 5; break;  
    case 2: $x = $x - 2; break;  
    case 3: $x = $x - 9; break;  
    default: $x = $x + 3; $y++;  
}
```

the statement `$y++` is executed only when variable `$a` contains a value other than 1, 2, or 3.

19.3 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the next PHP script for three different executions. Then, design the corresponding flowchart.

The input values for the three executions are (i) 1, (ii) 3, and (iii) 250.

```
<?php  
$a = trim(fgets(STDIN));  
  
$x = 0;  
$y = 0;  
switch ($a) {  
    case 1: $x = $x + 5; $y = $y + 5; break;  
    case 2: $x = $x - 2; $y--; break;  
    case 3: $x = $x - 9; $y = $y + 3; break;  
    default: $x = $x + 3; $y++;  
}  
  
echo $x, $y;  
?>
```

2. Create a trace table to determine the values of the variables in each step of the next PHP script for three different executions. Then, design the corresponding flowchart.

The input values for the three executions are (i) 10, 2, 5; (ii) 5, 2, 3; and (iii) 4, 6, 2.

```
<?php  
$a = trim(fgets(STDIN));  
$x = trim(fgets(STDIN));  
$y = trim(fgets(STDIN));
```

```
switch ($a) {  
    case 10:  
        $x = $x % 2;  
        $y = pow($y, 2);  
        break;  
    case 3:  
        $x = $x * 2;  
        $y = $y--;  
        break;  
    case 5:  
        $x = $x + 4;  
        $y += 7;  
        break;  
    default:  
        $x -= 3;  
        $y++;  
}  
  
echo $x, $y;  
?>
```

3. Write a PHP script that prompts the user to enter the name of a month, and then displays the corresponding number (1 for January, 2 for February, and so on). If the value entered is invalid, an error message should be displayed. The script should function properly even when names are given in capitals. For example, the script should function properly either for “January” or “JANUARY”.
4. Design a flowchart and write the corresponding PHP script that displays the following menu:
 1. Convert Miles to Yards
 2. Convert Miles to Feet
 3. Convert Miles to Inches

It then prompts the user to enter a choice (of 1, 2, or 3) and a distance in miles. Then, it calculates and displays the required value. However, if the choice entered is invalid, an error message should be displayed. It is given that

- 1 mile = 1760 yards
- 1 mile = 5280 feet
- 1 mile = 63360 inches

5. Roman numerals are shown in the following table.

Number	Roman Numeral
1	I
2	II
3	III
4	IV
5	V
6	VI
7	VII
8	VIII
9	IX
10	X

Write a PHP script that prompts the user to enter a Roman numeral between I and X, and then displays the corresponding number. However, if the choice entered is invalid, an error message should be displayed.

6. An online CD shop awards points to its customers based on the total number of audio CDs purchased each month. The points are awarded as follows:
- If the customer purchases 1 CD, he or she is awarded 3 points.
 - If the customer purchases 2 CDs, he or she is awarded 10 points.
 - If the customer purchases 3 CDs, he or she is awarded 20 points.
 - If the customer purchases 4 CDs or more, he or she is awarded 45 points.

Write a PHP script that prompts the user to enter the total number of CDs that he or she has purchased in a month, and then displays the number of points awarded. Assume that the user enters values greater than 0.

7. Write a PHP script that prompts the user to enter his or her name, and then displays "Good morning NN" or "Good evening NN" or "Good night NN", where NN is the name of the user. The message to be displayed must be chosen randomly.
8. Write a PHP script that lets the user enter a word such as "zero", "one" or "two", and then converts it into the corresponding digit, such as 0, 1, or 2. This should be done for the numbers 0 to 9. Display "I don't know this number!" when the user enters an unknown.
9. The Beaufort¹ scale is an empirical measure that relates wind speed to observed conditions on land or at sea. Write a PHP script that prompts the

¹ Francis Beaufort (1774–1857) was an Irish hydrographer and officer in Britain's Royal Navy. He is the inventor of the Beaufort wind force scale.

user to enter the Beaufort number, and then displays the corresponding description from the following table. However, if the number entered is invalid, an error message should be displayed.

Beaufort Number	Description
0	Calm
1	Light air
2	Light breeze
3	Gentle breeze
4	Moderate breeze
5	Fresh breeze
6	Strong breeze
7	Moderate gale
8	Gale
9	Strong gale
10	Storm
11	Violent storm
12	Hurricane force

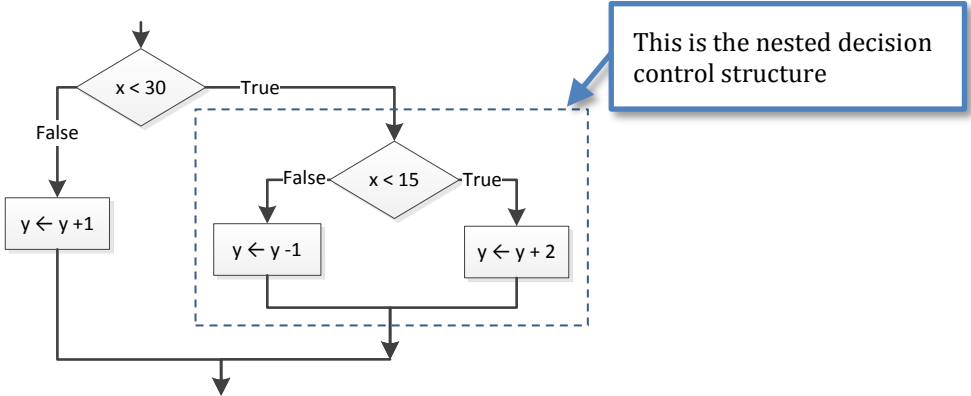
Chapter 20

Nested Decision Control Structures

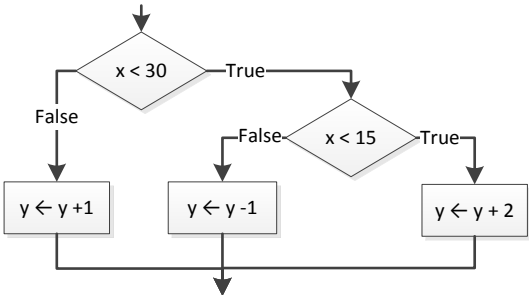
20.1 What are Nested Decision Control Structures?

Nested decision control structures are decision control structures that are “nested” (enclosed) within a path of another decision control structure. This means that one if statement can nest (enclose) another if statement, and the nested if statement can nest (enclose) another if statement, and so on. There are no practical limitations to how deep this nesting can go. As long as the syntax rules are not violated, you can nest as many decision control structures as you wish. For practical reasons however, as you move to three or four levels of nesting, the entire decision structure becomes very complex and difficult to understand. So, try to keep your code as simple as possible by breaking large nested decision control structures into multiple smaller ones, or by using other types of decision control structures.

An example of a nested decision control structure is shown here.



This can be rearranged to become



and the PHP script is shown here.

```
if ($x < 30) {  
    if ($x < 15) {  
        $y = $y + 2;  
    }  
    else {  
        $y = $y + 1;  
    }  
}
```

```

        $y--;
    }
}
else {
    $y++;
}

```

You can even nest case decision structures inside dual-alternative decision structures.

file_20.1.php

```

<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));

if ($x < 1 || $x > 3) {
    echo "Invalid Number";
}
else {
    switch ($x) {
        case 1:
            echo "1st choice selected";
            break;
        case 2:
            echo "2nd choice selected";
            break;
        case 3:
            echo "3rd choice selected";
            break;
    }
}
?>

```

This is a nested
case decision structure

Notice: Please note that keyword `default` is missing from the `switch` statement.

In general, you can nest any control structure inside any other control structure as long as you keep them syntactically and logically correct. However, you should always try to keep your script as simple as possible because complex code may lead to invalid results.

Exercise 20.1-1 Trace Tables and Nested Decision Control Structures

Create a trace table to determine the values of the variables in each step of the next PHP script for three different executions.

The input values for the three executions are (i) 13, (ii) 18, and (iii) 30.

file_20.1-1.php

```
<?php
    $x = trim(fgets(STDIN));
    $y = 10;

    if ($x < 30) {
        if ($x < 15) {
            $y = $y + 2;
        }
        else {
            $y--;
        }
    }
    else {
        $y++;
    }

    echo $y;

?>
```

Solution

i. For the input value of 13, the trace table looks like this.

Step	Statement	Notes	\$x	\$y
1	\$x = trim(fgets(STDIN));	User enters the value 13	13	?
2	\$y = 10;		13	10
3	if (\$x < 30)	This evaluates to true		
4	if (\$x < 15)	This evaluates to true		
5	\$y = \$y + 2;		13	12
6	echo \$y;	The value 12 is displayed		

ii. For the input value of 18, the trace table looks like this.

Step	Statement	Notes	\$x	\$y
1	\$x = trim(fgets(STDIN));	User enters the value 18	18	?
2	\$y = 10;		18	10
3	if (\$x < 30)	This evaluates to true		
4	if (\$x < 15)	This evaluates to false		

5	<code>\$y--;</code>		18	9
6	<code>echo \$y;</code>	The value 9 is displayed		

iii. For the input value of 30, the trace table looks like this.

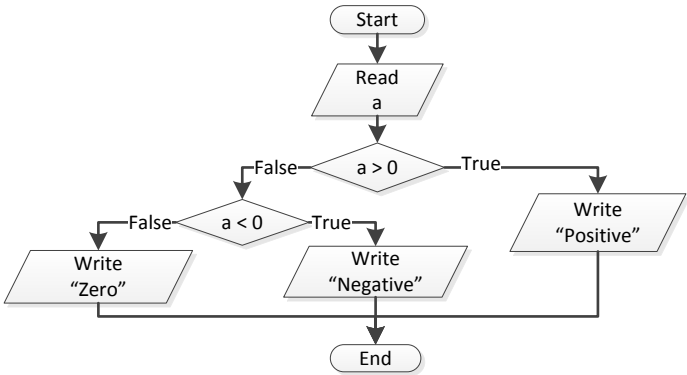
Step	Statement	Notes	\$x	\$y
1	<code>\$x = trim(fgets(STDIN));</code>	User enters the value 30	30	?
2	<code>\$y = 10;</code>		30	10
3	<code>if (\$x < 30)</code>	This evaluates to false		
4	<code>\$y++;</code>		30	11
5	<code>echo \$y;</code>	The value 11 is displayed		

Exercise 20.1-2 *Positive, Negative or Zero?*

Design a flowchart and write the corresponding PHP script that lets the user enter a number from the keyboard and then displays the messages “Positive”, “Negative”, or “Zero” depending on whether the given value is greater than, less than, or equal to zero.

Solution

The flowchart is shown here.



This PHP script can be written using either a nested decision control structure or a multiple-alternative decision structure. Let’s try them both!

First approach – Using a nested decision control structure

file_20.1-2a.php

```
<?php
$a = trim(fgets(STDIN));

if ($a > 0) {
    echo "Positive";
}
```

```

}
else {
    if ($a < 0) {
        echo "Negative";
    }
    else {
        echo "Zero";
    }
}
?>

```

Second approach – Using a multiple-alternative decision structure

file_20.1-2b.php

```

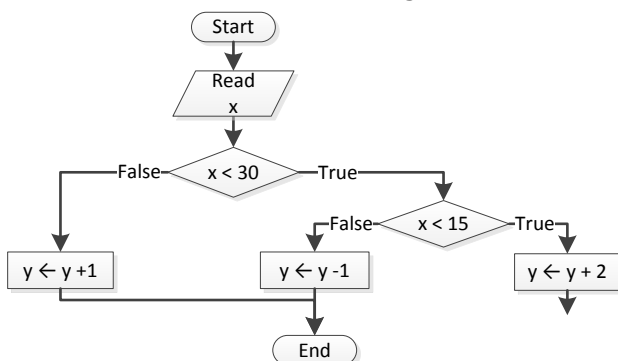
<?php
$a = trim(fgets(STDIN));

if ($a > 0) {
    echo "Positive";
}
elseif ($a < 0) {
    echo "Negative";
}
else {
    echo "Zero";
}
?>

```

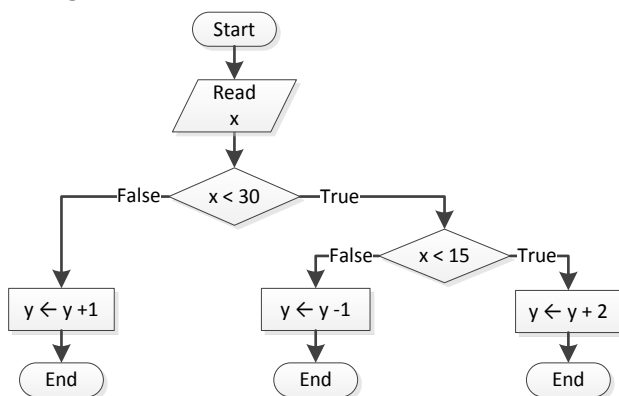
20.2 A Mistake That You Will Probably Make!

In flowcharts, a very common mistake that novice programmers make is to leave some paths unconnected, as shown in the following flowchart.

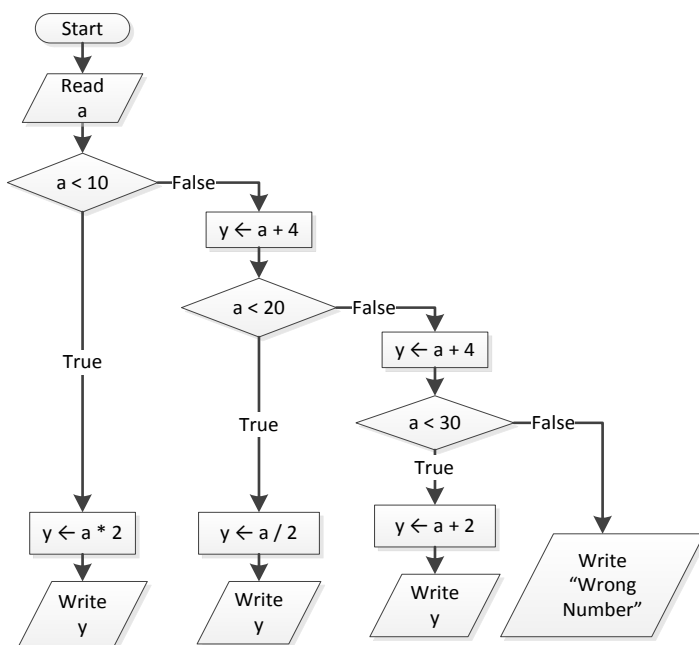


Please keep in mind that every path tries to reach the end of the algorithm, thus you cannot leave any of them unconnected.

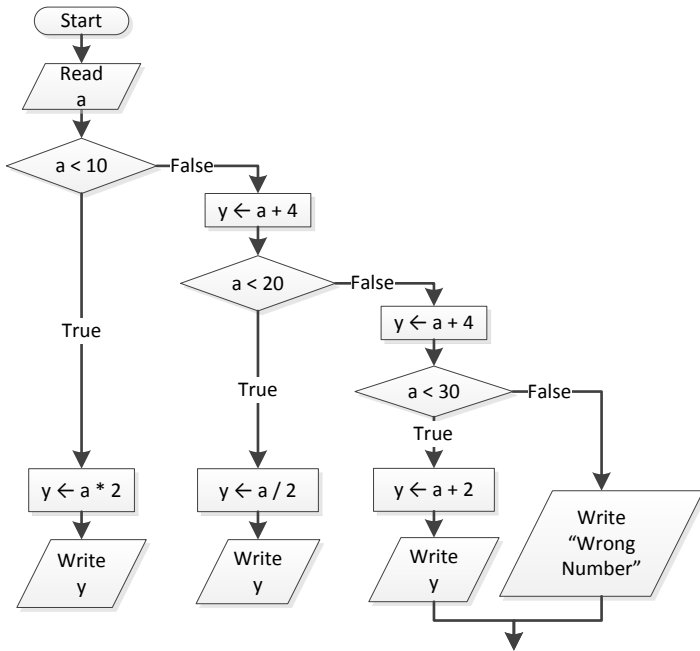
On the other hand, try to avoid flowcharts that use many End symbols, as shown below, since these algorithms are difficult to read and understand.



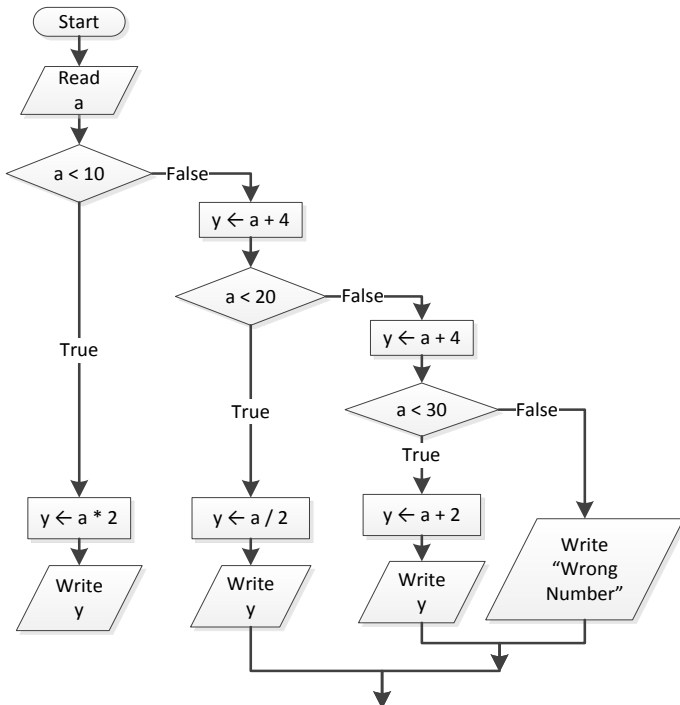
So, let's say that you are in the middle of designing a flowchart, and you want to start closing all of its decision control structures.



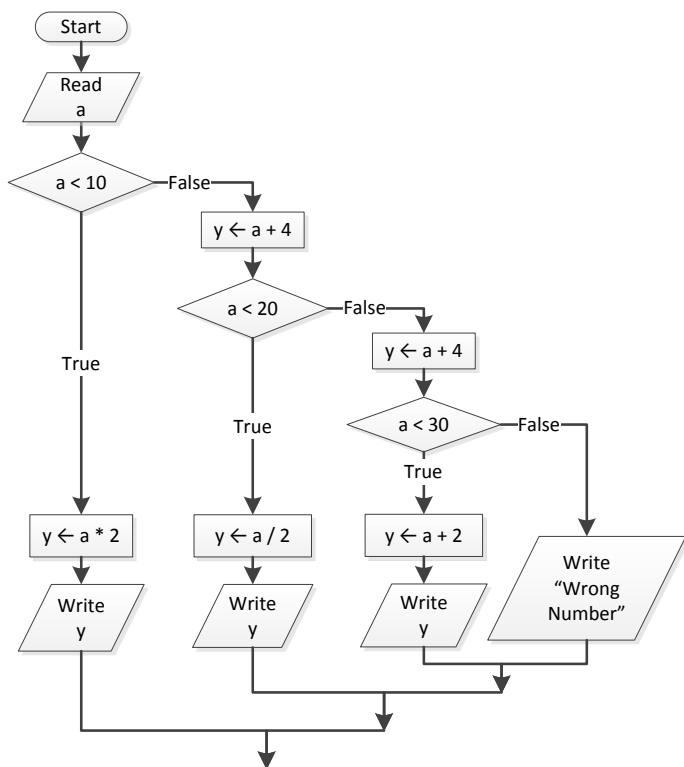
Just remember that the decision control structure that opens last should be the first one to close! In this example, the last decision control structure is the one that evaluates the expression $a < 30$. This is the first one that you need to close, as shown in the flowchart that follows.



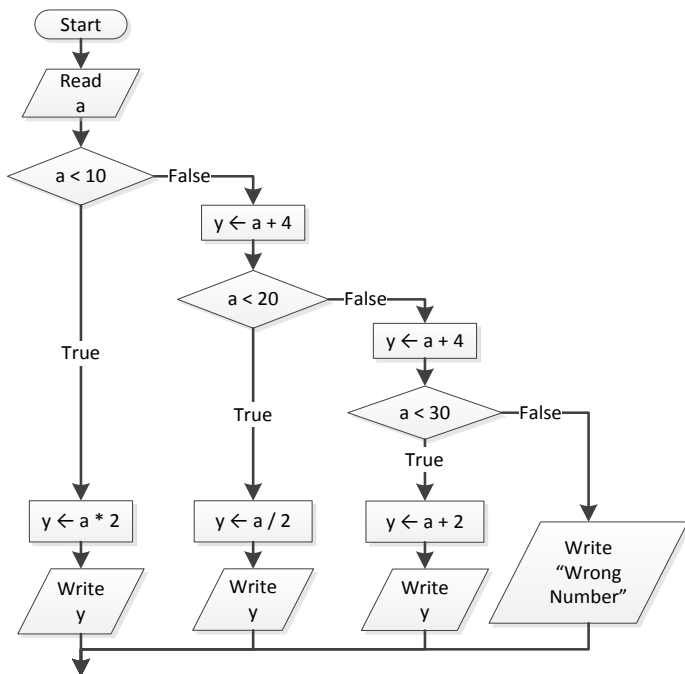
Next, you need to close the second to last decision control structure as shown here.



And finally, you need to close the third to last decision control structure as shown here.



The last flowchart can be rearranged to become like the one that follows.



20.3 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Nesting of decision control structures describes a situation in which one or more than one paths of a decision control structure enclose other decision control structures.
2. Nesting level can go as deep as the programmer wishes.
3. When a problem can be solved using either a case decision structure or nested decision control structures, the second option is better because the script becomes more readable.
4. It is possible to nest a multiple-alternative decision structure within a case decision structure, but not the opposite.
5. When designing flowcharts with nested decision control structures, the decision control structure that opens last should be the first one to close.

20.4 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the next PHP script for four different executions.

The input values for the four executions are (i) 20, 1; (ii) 20, 3; (iii) 12, 8; and (iv) 50, 0.

```
<?php
```

```
$x = trim(fgets(STDIN));
$y = trim(fgets(STDIN));

if ($x < 30) {
    switch ($y) {
        case 1:
            $x = $x % 3;
            $y = 5;
            break;
        case 2:
            $x = $x * 2;
            $y = 2;
            break;
        case 3:
            $x = $x + 5;
            $y += 3;
            break;
        default:
            $x -= 2;
            $y++;
    }
}
else {
    $y++;
}

echo $x, $y;

?>
```

2. Create a trace table to determine the values of the variables in each step of the next PHP script for four different executions.

The input values for the four executions are (i) 60, 25; (ii) 50, 8; (iii) 20, 15; and (iv) 10, 30.

```
<?php
$x = trim(fgets(STDIN));
$y = trim(fgets(STDIN));

if (($x + $y) / 2 <= 20) {
    if ($y < 10) {
        $x = $x % 3;
        $y += 2;
    }
}
```

```
    }  
    elseif ($y < 20) {  
        $x = $x * 5;  
        $y += 2;  
    }  
    else {  
        $x = $x - 2;  
        $y += 3;  
    }  
}  
else {  
    if ($y < 15) {  
        $x = $x % 4;  
        $y = 2;  
    }  
    elseif ($y < 23) {  
        $x = $x % 2;  
        $y -= 2;  
    }  
    else {  
        $x = 2 * $x + 5;  
        $y += 1;  
    }  
}  
  
echo $x, $y;  
?>
```

3. Write a PHP script that prompts the user to enter the lengths of three sides of a triangle, and then determines whether or not the given numbers can be lengths of the three sides of a triangle. If the lengths are not valid, a corresponding message should be displayed; otherwise the script should further determine whether the triangle is
- equilateral
Hint: In an equilateral triangle, all sides are equal.
 - right (or right-angled)
Hint: Use the Pythagorean theorem.
 - not special

Hint: In any triangle, the length of each side is less than the sum of the lengths of the other two sides.

4. Inside an automated teller machine (ATM) there are notes of \$10, \$5, and \$1. Write a PHP script to emulate the way this ATM works. At the beginning, the machine prompts the user to enter the four-digit PIN and then checks for PIN validity (assume “1234” as the valid PIN). If given PIN is correct, the script should prompt the user to enter the amount of money (an integer value) that he or she wants to withdraw and finally it displays the least number of notes the ATM should dispense. For example, if the user enters an amount of \$36, the script should display “3 notes of \$10, 1 note of \$5, and 1 note of \$1”. Moreover, if the user enters a wrong PIN, the machine will allow him or her two retries. If the user enters an incorrect PIN all three times, the message “PIN locked” should be displayed and the script should end.
5. Write a PHP script that prompts the user to enter two values, one for temperature and one for wind speed. If the temperature is above 75 degrees Fahrenheit, the day is considered hot, otherwise it is cold. If the wind speed is above 12 miles per hour, the day is considered windy, otherwise it is not windy. The script should display one single message, depending on values given. For example, if a user enters 60 for temperature and 10 for wind speed, the script should display “The day is cold and not windy”.

Chapter 21

Tips and Tricks with Decision Control Structures

21.1 Introduction

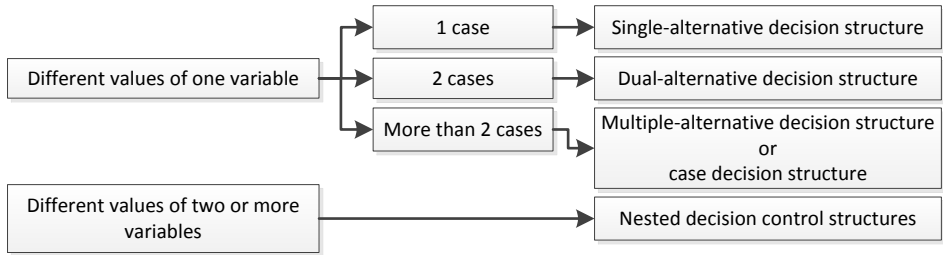
This chapter is dedicated to teaching you some useful tips and tricks that can help you write “better” code. You should always keep them in mind when you design your own algorithms, or even your own PHP scripts.

These tips and tricks can help you increase your code’s readability and help make the code shorter or even faster. Of course there is no single perfect methodology because on one occasion the use of a specific tip or trick may help, but on another occasion the same tip or trick may have exactly the opposite result. Most of the time, code optimization is a matter of programming experience.

Remember! *Smaller algorithms are not always the best solution to a given problem. In order to solve a specific problem, you might write a very short algorithm that unfortunately proves to consume a lot of CPU time. On the other hand, you may solve the same problem with another algorithm which, even though it seems longer, calculates the result much faster.*

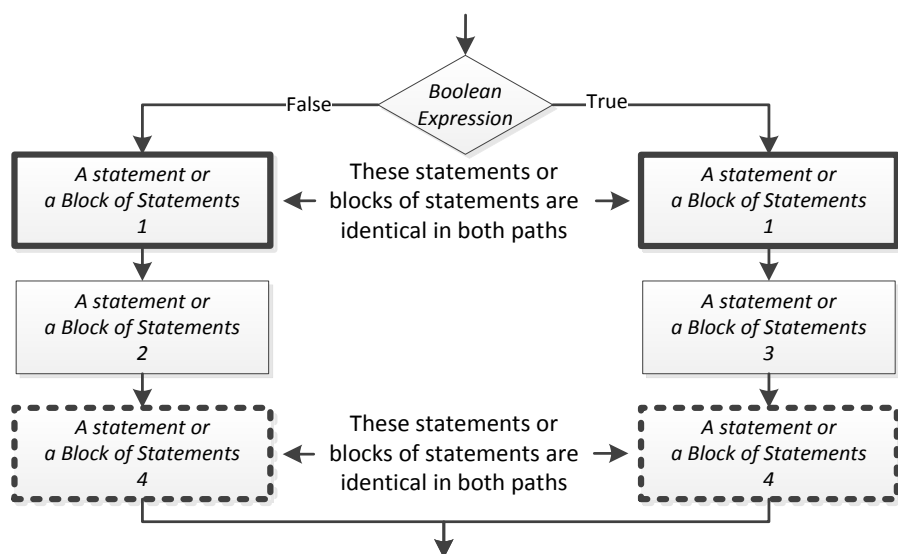
21.2 Choosing a Decision Control Structure

The following diagram can help you decide which decision control structure is a better choice for a given problem depending on the number of variables checked.

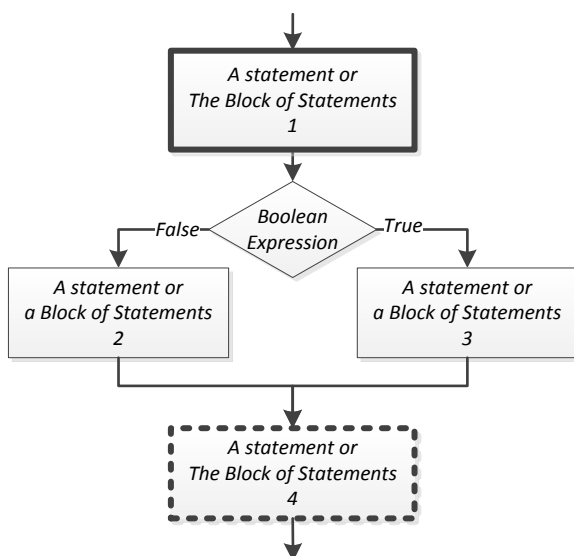


21.3 Streamlining the Decision Control Structure

Look carefully at the following flowchart fragment given in general form.



As you can see, two identical statements or blocks of statements exist at the beginning and two other identical statements or blocks of statements exist at the end of both paths of the dual-alternative decision structure. This means that, regardless of the result of the Boolean expression, these statements are executed either way. Thus, you can simply move them outside and (respectively) right before and right after the dual-alternative decision structure, as shown in this equivalent structure.

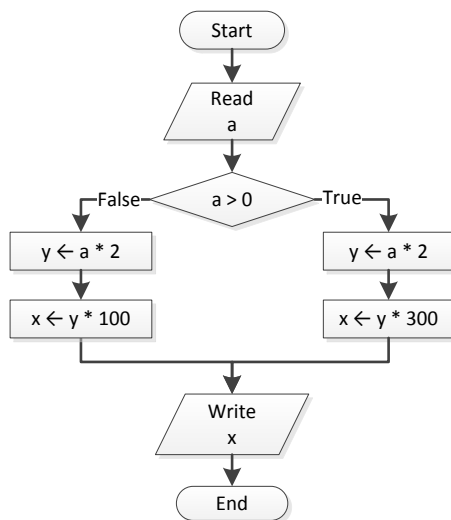


The same tip can be applied to any decision control structure, as long as an identical statement or block of statements exists in all paths. Of course, there are also cases where this tip cannot be applied.

Are you still confused? Next, you will find some exercises that can help you to understand better.

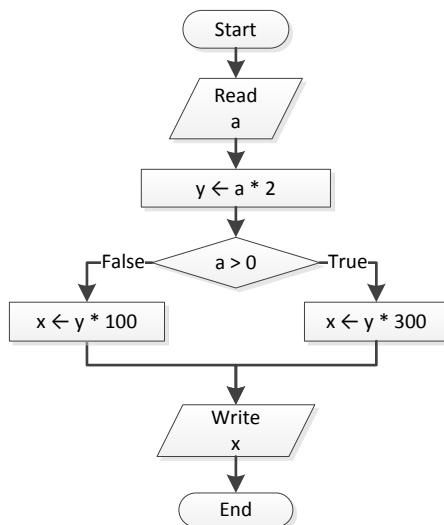
Exercise 21.3-1 “Shrinking” the Algorithm

Redesign the following flowchart using fewer statements.



Solution

As you can see, the statement $y \leftarrow a * 2$ exists in both paths of the dual-alternative decision structure. This means that, regardless of the result of the Boolean expression, this statement is executed either way. Therefore, you can simply move the statement outside and right before the dual-alternative decision structure, as shown in the flowchart that follows.



Exercise 21.3-2 “Shrinking” the PHP Script

Rewrite the following PHP script using fewer statements.

```

<?php
$a = trim(fgets(STDIN));

if ($a > 0) {
    $y = $a * 4;
    echo y;
}
else {
    $y = $a * 3;
    echo $y;
}
?>
  
```

Solution

As you can see, the statement `echo $y` exists in both paths of the dual-alternative decision structure. This means that, regardless of the result of the Boolean expression, this statement is executed either way. Therefore, you can simply move the statement outside and right after the dual-alternative decision structure, as shown here.

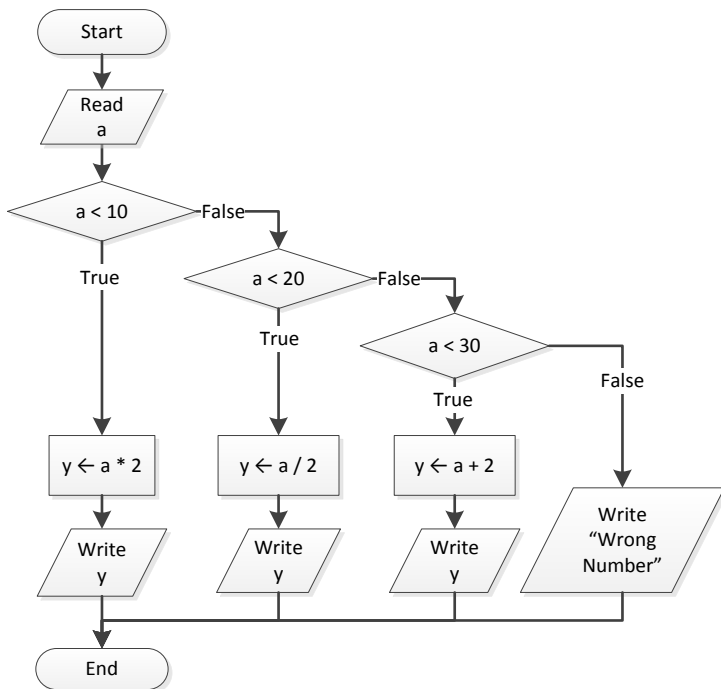
```

<?php
$a = trim(fgets(STDIN));
  
```

```
if ($a > 0) {  
    $y = $a * 4;  
}  
else {  
    $y = $a * 3;  
}  
  
echo $y;  
?>
```

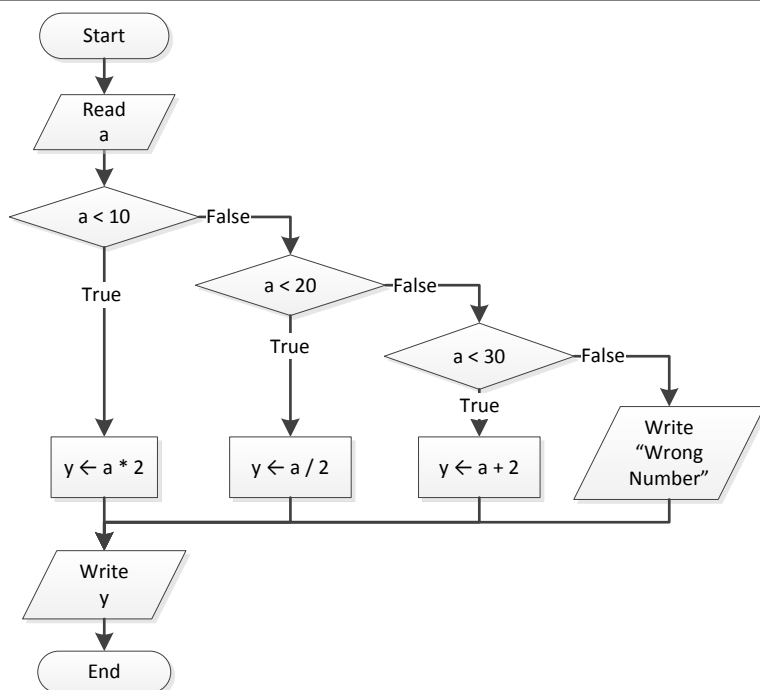
Exercise 21.3-3 “Shrinking” the Algorithm

Redesign the following flowchart using fewer statements.



Solution

In this case, nothing special can be done. If you try to move the `Write y` statement outside of the multiple-alternative decision structure, the resulting flowchart that follows is definitely **not** equivalent to the initial one.

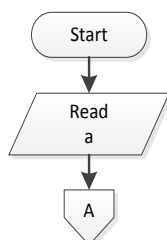


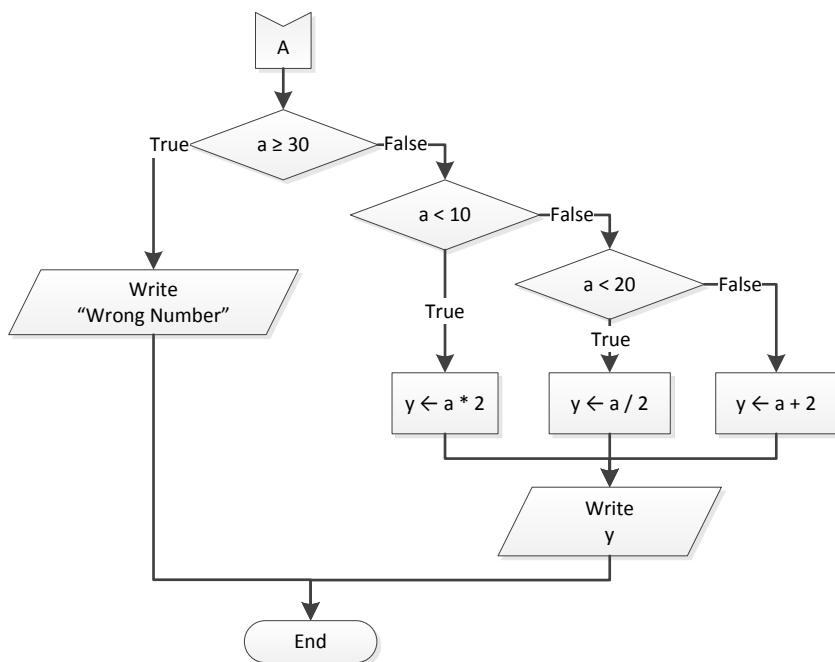
This is because of the last path on the right side which, in the initial flowchart, didn't include the `Write y` statement.

Examine both flowcharts to see whether they produce the same result. For example, suppose a user enters a wrong number. In both flowcharts, the flow of execution goes to the `Write "Wrong Number"` statement. After that, the initial flowchart executes no other statements but on the contrary, the second flowchart executes an extra `Write y` statement.

Remember! You cannot move a statement or block of statements outside of a decision control structure if it does not exist in all paths.

You may now wonder whether there is any other way to move the `Write y` statement outside of the multiple-alternative decision structure. The answer is “yes,” but you need to slightly rearrange the flowchart. You need to completely remove the last path on the right and use a brand new decision control structure in the beginning to check whether or not the given number is wrong. One possible solution is shown here.





and the PHP script is

```
<?php
```

```
$a = trim(fgets(STDIN));
```

```
if ($a >= 30) {
```

```
    echo "Wrong Number";
```

```
}
```

```
else {
```

```
    if ($a < 10) {
```

```
        $y = $a * 2;
```

```
    }
```

```
    elseif ($a < 20) {
```

```
        $y = $a / 2;
```

```
    }
```

```
    else {
```

```
        $y = $a + 2;
```

```
    }
```

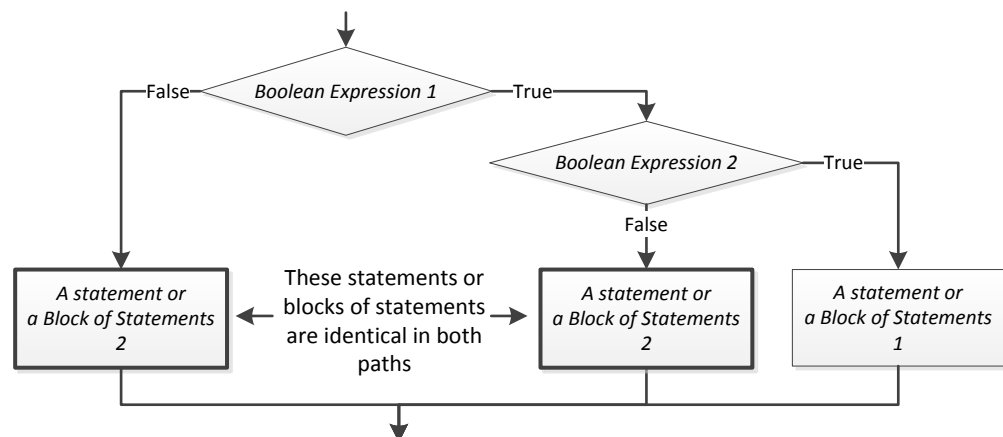
```
    echo $y;
```

```
}
```

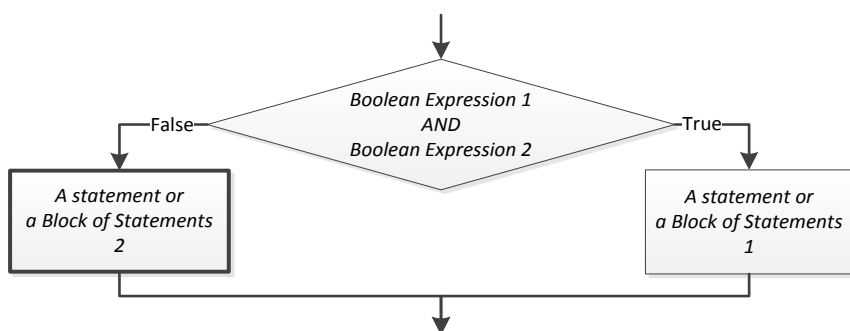
```
?>
```

21.4 Logical Operators – to Use, or not to Use: That is the Question!

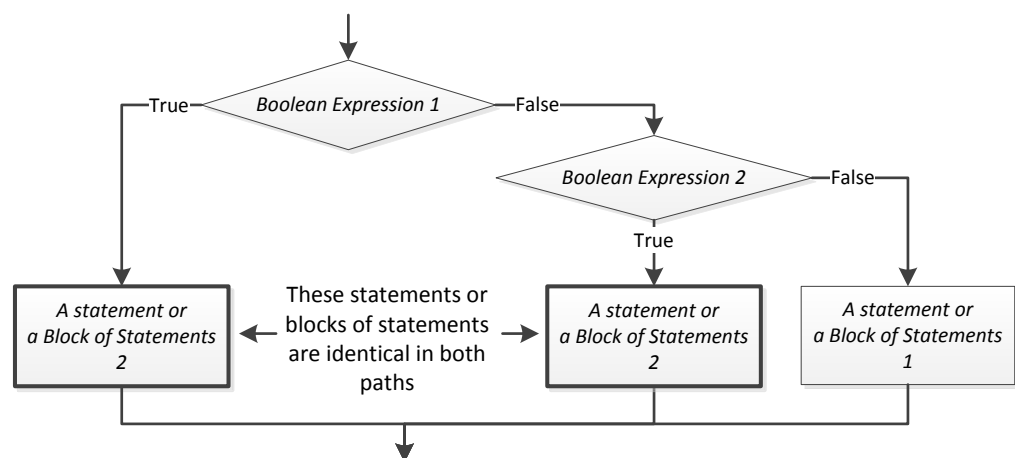
There are some cases in which you can use a logical operator instead of nested decision control structures, and this can lead to increased readability. Take a look at the following flowchart fragment given in general form.



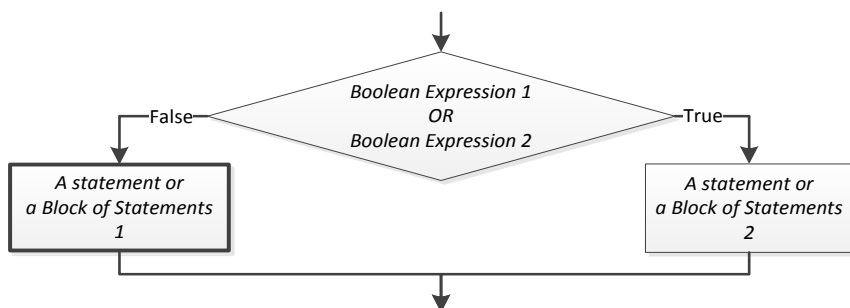
As you can see, the statement or block of statements 1 is executed only when **both** Boolean expressions evaluate to `true`. The statement or block of statements 2 is executed in all other cases. Therefore, this flowchart fragment can be redesigned using the AND (&&) logical operator.



Now, let's take a look at another flowchart fragment given in general form.



In this flowchart fragment, the statement or block of statements 2 is executed when **either** Boolean expression 1 evaluates to **true** **or** Boolean expression 2 evaluates to **true**. Therefore, you can redesign this flowchart fragment using the OR (||) logical operator as shown here.



Notice: Obviously, these methodologies can be adapted to be used on nested decision control structures as well.

Exercise 21.4-1 Rewriting the Code

Rewrite the following PHP script using logical operators.

```

<?php
    $today = trim(fgets(STDIN));
    $name = trim(fgets(STDIN));

    if ($today == "February 16") {
        if ($name == "Loukia") {

```

```
        echo "Happy Birthday!!!";
    }
    else {
        echo "No match!";
    }
}
else {
    echo "No match!";
}
?>
```

Solution

The `echo "Happy Birthday!!!"` statement is executed only when **both** Boolean expressions evaluate to true. The statement `echo "No match!"` is executed in all other cases. Therefore, you can rewrite the PHP script using the AND (`&&`) logical operator.

```
<?php
$today = trim(fgets(STDIN));
$name = trim(fgets(STDIN));

if ($today == "February 16" && $name == "Loukia") {
    echo "Happy Birthday!!!";
}
else {
    echo "No match!";
}
?>
```

Exercise 21.4-2 *Rewriting the Code*

Rewrite the following PHP script using logical operators.

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$y = 0;

if ($a > 10) {
    $y++;
}
```



```
elseif ($b > 20) {  
    $y++;  
}  
else {  
    $y--;  
}  
  
echo $y;  
?>
```

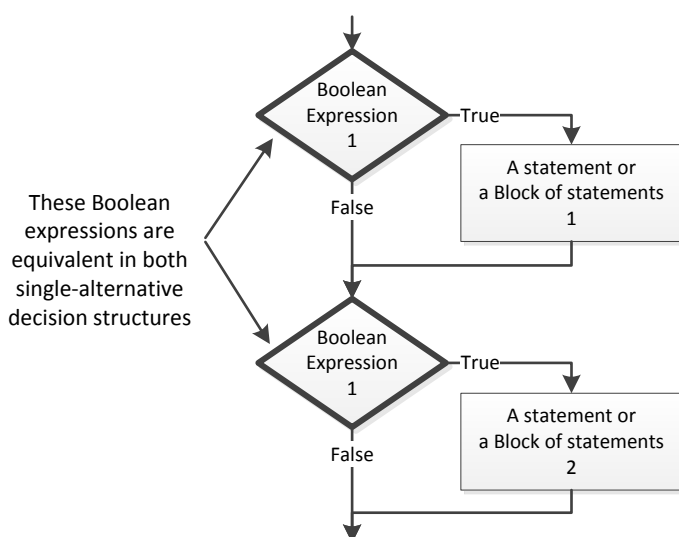
Solution

The `$y++` statement is executed when **either** variable `$a` is greater than 10 **or** variable `$b` is greater than 20. Therefore, you can rewrite the PHP script using an OR (`||`) logical operator.

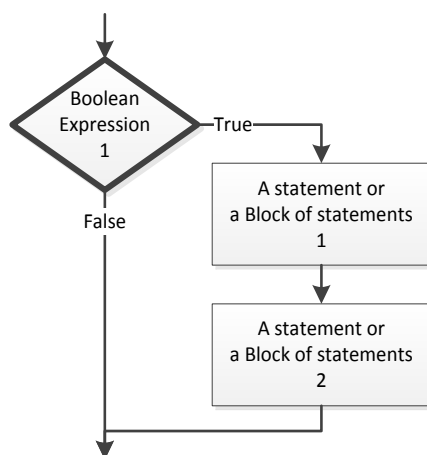
```
<?php  
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
  
$y = 0;  
  
if ($a > 10 || $b > 20) {  
    $y++;  
}  
else {  
    $y--;  
}  
  
echo $y;  
?>
```

21.5 Merging Two or More Single-Alternative Decision Structures

Many times, an algorithm contains two or more single-alternative decision structures in a row which actually evaluate the same Boolean expression. An example is shown next.



When a situation like this occurs, you can just merge all single-alternative decision structures to a single one as shown here.



Remember! The single-alternative decision structures need to be adjacent to each other. If any statement exists between them, you can't merge them unless you are able to move this statement to somewhere else in your code.

Exercise 21.5-1 Merging the Decision Control Structures

In the following PHP script, merge the single-alternative decision structures.

```

<?php
$a = trim(fgets(STDIN));

```

```
if ($a > 0) {  
    echo "Hello ";  
}  
  
if ($a > 0) {  
    echo "Hermes";  
}  
?>
```

Solution

The first and second decision control structures are evaluating exactly the same Boolean expressions, so they can simply be merged into a single one.

The PHP script becomes

```
<?php  
$a = trim(fgets(STDIN));  
  
if ($a > 0) {  
    echo "Hello ";  
    echo "Hermes";  
}  
?>
```

Exercise 21.5-2 Merging the Decision Control Structures

In the following PHP script, merge as many single-alternative decision structures as possible.

```
<?php  
$a = trim(fgets(STDIN));  
  
$y = 0;  
  
if ($a > 0) {  
    $y++;  
}  
  
$b = trim(fgets(STDIN)); //This statement is not affected by the  
                          //previous decision control structure  
                          //and does not affect the next one  
  
if (!($a <= 0)) {
```

```
    echo "Hello Hera";
}

$a++;                                //The previous and next decision control
                                    //structures are dependent upon this
                                    //statement

if ($a > 0) {
    echo "Hallo Welt";
}

echo $b;

?>
```

Solution

If you take a closer look at the PHP script, it becomes clear that the first and second decision control structures are actually evaluating exactly the same Boolean expression. Negation of `$a > 0` results in `$a <= 0`, and a second negation of `$a <= 0` (using the NOT operator this time) results in `!($a <= 0)`. Thus, `$a > 0` is in fact equivalent to `!($a <= 0)`.

Remember! *Two negations result in an affirmative.*

However, between the first and second decision control structures there is the statement `$b = trim(fgets(STDIN))`, which prevents you from merging them into a single one. Fortunately, this statement can be moved to the beginning of the script since it doesn't really affect the rest flow of execution.

On the other hand, between the second and third decision control structures there is the statement `$a++`, which also prevents you from merging; however, this statement cannot be moved anywhere else because it does affect the rest of the flow of execution (the second and third decision control structures are dependent upon this statement). Thus, the third decision control structure cannot be merged with the first and second ones!

The final PHP script looks like this.

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$y = 0;

if ($a > 0) {
    $y++;
    echo "Hello Hera";
```

```
}

$a++;

if ($a > 0) {
    echo "Hallo Welt";
}

echo $b;

?>
```

21.6 Replacing Two Single-Alternative Decision Structures with a Dual-Alternative One

Take a look at the next example.

```
if ($a > 40) {
    //do something
}

if ($a <= 40) {
    //do something else
}
```

The first decision control structure evaluates variable `$a` to test if it is bigger than 40, and right after that, a second decision control structure evaluates the same variable again to test if it is less than or equal to 40!

This is a very common “mistake” that novice programmers make. They use two single-alternative decision structures even though one dual-alternative decision structure can accomplish the same thing.

The previous example can be rewritten using only one dual-alternative decision structure, as shown here.

```
if ($a > 40) {
    //do something
}
else {
    //do something else
}
```

Even though both examples are absolutely correct and work perfectly well, the second alternative is better. The CPU needs to evaluate only one Boolean expression, which results in faster execution time.

Notice: The two single-alternative decision structures must be adjacent to each other. If any statement exists between them, you can't "merge" them (that is, replace them with a dual-alternative decision structure) unless you are able to move this statement to somewhere else in your code.

Exercise 21.6-1 "Merging" the Decision Control Structures

In the following PHP script, "merge" as many single-alternative decision structures as possible.

```
<?php
    $a = trim(fgets(STDIN));

    $y = 0;

    if ($a > 0) {
        $y++;
    }

    $b = trim(fgets(STDIN));           //This statement is not affected by the
                                       //previous decision control structure
                                       //and does affect the next one.
                                       //Thus, it can be moved to the beginning
                                       //of the script

    if (!($a > 0)) {
        echo "Hello Zeus";
    }

    if ($y > 0) {                       //Unfortunately this decision control
        echo $y + 5;                   // structure can't be merged with
    }                                  //the next one...

    $y++;                               //... because of this statement

    if ($y <= 0) {
        echo $y + 12;
    }

?>
```

Solution

The first decision control structure evaluates variable `$a` to test if it is greater than zero, and just right after that the second decision control structure evaluates variable

`$a` again to test if it is not greater than zero. Even though there is the statement `$b = trim(fgets(STDIN))` between them, this statement can be moved somewhere else because it doesn't really affect the rest of the flow of execution.

On the other hand, between the third and fourth decision control structures there is the statement `$y++` which also prevents you from merging. However, this statement cannot be moved anywhere else because it does affect the rest of the flow of execution (the third and fourth decision control structures are dependent upon this statement). Therefore, the third and fourth decision control structures cannot be merged!

The final PHP script becomes

```
<?php
    $a = trim(fgets(STDIN));
    $b = trim(fgets(STDIN));

    $y = 0;

    if ($a > 0) {
        $y++;
    }
    else {
        echo "Hello Zeus";
    }

    if ($y > 0) {
        echo $y + 5;
    }

    $y++;

    if ($y <= 0) {
        echo $y + 12;
    }
?>
```

21.7 Put the Boolean Expressions Most Likely to be True First

Both the multiple-alternative and the case decision structure often need to check several Boolean expressions before deciding which statement or block of statements to execute. In the next decision control structure,

```
if (Boolean_Expression 1) {
    A statement or block of statements 1
}
```

```
elseif (Boolean_Expression 2) {  
    A statement or block of statements 2  
}
```

the program first tests if *Boolean_Expression 1* is true and if not, it tests if *Boolean_Expression 2* is true.

However, what if *Boolean_Expression 1* is false most of the time and *Boolean_Expression 2* is true most of the time? This means that time is wasted testing *Boolean_Expression 1*, which is usually false, before testing *Boolean_Expression 2*, which is usually true.

To make your scripts more efficient, you can put the Boolean expressions that are most likely to be true at the beginning, and the Boolean expressions that are most likely to be false at the end, using either a multiple-alternative or a case decision structure as shown here.

```
if (Boolean_Expression 2) {  
    A statement or block of statements 2  
}  
elseif (Boolean_Expression 1) {  
    A statement or block of statements 1  
}
```

Remember! Although this method may seem nonessential, every little bit of time that you save can add up to make your scripts run faster and more efficiently.

Exercise 21.7-1 *Rearranging the Boolean Expressions*

According to an ultra-high top secret report, America's favorite pets are dogs, with cats at second place, guinea pigs next, and parrots coming in last. In the following PHP script, rearrange the Boolean expressions to make the script run faster and more efficiently.

```
<?php  
echo "What is your favorite pet? ";  
$kind = trim(fgets(STDIN));  
  
switch ($kind) {  
    case "Parrots":  
        echo "It screeches!";  
        break;  
    case "Guinea pig":  
        echo "It squeaks";  
        break;  
    case "Dog":
```



```
    echo "It barks";  
    break;  
case "Cat":  
    echo "It meows";  
    break;  
}  
?>
```

Solution

For this top secret report, you can rearrange the PHP script to make it run a little bit faster for most of the cases.

```
<?php  
echo "What is your favorite pet? ";  
$kind = trim(fgets(STDIN));  
  
switch ($kind) {  
    case "Dog":  
        echo "It barks";  
        break;  
    case "Cat":  
        echo "It meows";  
        break;  
    case "Guinea pig":  
        echo "It squeaks";  
        break;  
    case "Parrots":  
        echo "It screeches!";  
        break;  
}  
?>
```

21.8 Converting a Case Decision Structure to a Multiple-Alternative Decision Structure, and Vice Versa

Converting a case decision structure to a multiple-alternative decision structure is quite easy. All you need to do is get the variable or the expression that the switch statement evaluates and use it in the multiple-alternative decision structure for all cases.

Case Decision Structure

```
switch (var_expr) {  
    case value-1:  
        block of statements 1  
        break;  
    case value-2:  
        block of statements 2  
        break;  
    case value-3:  
        block of statements 3  
        break;  
    .  
    .  
    .  
    case value-N:  
        block of statements N  
        break;  
    default:  
        block of statements N+1  
}
```

**Multiple-Alternative Decision Structure**

```
if (var_expr == value-1) {  
    block of statements 1  
}  
elseif (var_expr == value-2) {  
    block of statements 2  
}  
elseif (var_expr == value-3) {  
    block of statements 3  
}  
.  
.  
.  
elseif (var_expr == value-N) {  
    block of statements N  
}  
else {  
    block of statements N+1  
}
```

Notice: The opposite conversion (from multiple-alternative decision structure to case decision structure) is possible as long as the same variable or expression is evaluated for all paths. Moreover, this variable or expression should be tested in each path to determine if it is equal to the constant values value-1, value-2, and value-N respectively.

Notice: Although there is an alternative form of the switch statement in PHP, it is beyond the scope of this book and so is purposely not mentioned.

Exercise 21.8-1 Converting the PHP Script

Rewrite the following PHP script using the multiple-alternative decision structure.

```
<?php  
echo "What is your name? ";  
$name = trim(fgets(STDIN));  
  
switch ($name) {  
    case "John":  
        echo "You are my cousin!";
```

```
        break;
    case "Aphrodite":
        echo "You are my sister!";
        break;
    case "Loukia":
        echo "You are my mom!";
        break;
    default:
        echo "Sorry, I don't know you.";
    }
?>
```

Solution

The multiple-alternative decision structure should evaluate variable `$name` for all paths.

```
<?php
echo "What is your name? ";
$name = trim(fgets(STDIN));

if ($name == "John") {
    echo "You are my cousin!";
}
elseif ($name == "Aphrodite") {
    echo "You are my sister!";
}
elseif ($name == "Loukia") {
    echo "You are my mom!";
}
else {
    echo "Sorry, I don't know you.";
}
?>
```

Exercise 21.8-2 *Converting the PHP Script*

Rewrite the following PHP script using the case decision structure.

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));
```

```
if ($x == 1) {  
    echo "One is good.";  
}  
elseif ($x == 2) {  
    echo "Two are always better than one!";  
}  
elseif ($x == 3) {  
    echo "Three is superb!";  
}  
?  
>
```

Solution

In this script, the same variable is evaluated for all paths. Moreover, this variable is tested in each path to determine if it is equal to constant values 1, 2, or 3 respectively. The conversion is possible and is shown here.

```
<?php  
echo "Enter a number: ";  
$x = trim(fgets(STDIN));  
  
switch ($x) {  
    case 1:  
        echo "One is good.";  
        break;  
    case 2:  
        echo "Two are always better than one!";  
        break;  
    case 3:  
        echo "Three is superb!";  
        break;  
}  
?  
>
```

Notice: Please note that since the keyword `default` is missing, the statement `break` of the last case is actually redundant and can be omitted

Exercise 21.8-3 *Converting the PHP Script*

Rewrite the following PHP script using the case decision structure.

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));
echo "Enter a second number: ";
$y = trim(fgets(STDIN));

if ($x < 5) {
    echo "Variable x is less than 5";
}
elseif ($y == 3) {
    echo "Variable y equals to 3";
}
elseif ($x + $y == 24) {
    echo "The sum of x + y equals to 24";
}
elseif (sqrt($x) == 4) {
    echo "The square root of x equals to 4";
}
else {
    echo "None of the above";
}
?>
```

Solution

As you already know, the conversion from a multiple-alternative decision structure to a case decision structure is possible as long as the same variable or expression is evaluated for all paths. In this exercise, however, each path evaluates a different variable or expression. For that reason, the conversion to the classical form of the `switch` statement is not possible.

21.9 Converting a Multiple-Alternative Decision Structure to Nested Decision Control Structures, and Vice Versa

The next PHP statements, given in general form, can help you easily convert from a multiple-alternative decision structure to nested decision control structures, and vice versa.

Multiple-Alternative Decision Structure

```
if (Boolean Expr 1) {  
    block of statements 1  
}  
elseif (Boolean Expr 2) {  
    block of statements 2  
}  
elseif (Boolean Expr 3) {  
    block of statements 3  
}  
.  
.  
.  
elseif (Boolean Expr N) {  
    block of statements N  
}  
else {  
    block of statements N+1  
}
```



Nested Decision Control Structures

```
if (Boolean Expr 1) {  
    block of statements 1  
}  
else {  
    if (Boolean Expr 2) {  
        block of statements 2  
    }  
    else {  
        if (Boolean Expr 3) {  
            block of statements 3  
        }  
        .  
        .  
        .  
        else {  
            if (Boolean Expr N) {  
                block of statements N  
            }  
            else {  
                block of statements N+1  
            }  
        }  
    }  
}
```

Notice: The opposite conversion (from nested decision control structures to a multiple-alternative decision structure) is possible as long as the `else` keyword and the succeeding `if` statement (in the nested decision control structures) are side by side without any statements existing between them.

Exercise 21.9-1 Converting the PHP Script

Rewrite the PHP script using nested decision control structures.

```
<?php  
echo "Enter a number: ";  
$x = trim(fgets(STDIN));  
echo "Enter a second number: ";  
$y = trim(fgets(STDIN));
```

```
$result = $x * $y;

if ($x < 0 && $y < 0) {
    echo "The result of the multiplication of\n";
    echo "two negative numbers is positive. For example\n";
    echo "The result of ", $x, " * (" , $y, ") is +", $result;
}
elseif ($x < 0) {
    echo "The result of the multiplication of a negative\n";
    echo "with a positive number is negative. For example\n";
    echo "The result of ", $x, " * ", $y, " is ", $result;
}
elseif ($y < 0) {
    echo "The result of the multiplication of a positive\n";
    echo "with a negative number is negative. For example\n";
    echo "The result of ", $x, " * (" , $y, ") is ", $result;
}
else {
    echo "The result of the multiplication of\n";
    echo "two positive numbers is positive. For example\n";
    echo "The result of ", $x, " * ", $y, " is +", $result;
}

?>
```

Solution

Using what you learned so far, the PHP script becomes

```
<?php
    echo "Enter a number: ";
    $x = trim(fgets(STDIN));
    echo "Enter a second number: ";
    $y = trim(fgets(STDIN));

    $result = $x * $y;

    if ($x < 0 && $y < 0) {
        echo "The result of the multiplication of\n";
        echo "two negative numbers is positive. For example\n";
        echo "The result of ", $x, " * (" , $y, ") is +", $result;
    }
}
```

```
else {
    if ($x < 0) {
        echo "The result of the multiplication of a negative\n";
        echo "with a positive number is negative. For example\n";
        echo "The result of ", $x, " * ", $y, " is ", $result;
    }
    else {
        if ($y < 0) {
            echo "The result of the multiplication of a positive\n";
            echo "with a negative number is negative. For example\n";
            echo "The result of ", $x, " * (", $y, ") is ", $result;
        }
        else {
            echo "The result of the multiplication of\n";
            echo "two positive numbers is positive. For example\n";
            echo "The result of ", $x, " * ", $y, " is +", $result;
        }
    }
}
?>
```

Exercise 21.9-2 *Converting the PHP Script*

Rewrite the PHP script using a multiple-alternative decision structure.

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));
echo "Enter a second number: ";
$y = trim(fgets(STDIN));

if ($x > 5) {
    echo "Variable x is greater than 5";
}
else {
    $x = $x + $y;
    echo "Hello Zeus!";
    if ($x == 3) {
        echo "Variable x equals to 3";
    }
    else {
```



```
$x = $x - $y;  
echo "Hello Olympians!";  
if ($x + $y == 24) {  
    echo "The sum of x + y equals to 24";  
}  
else {  
    echo "Nothing of the above";  
}  
}  
?  
?>
```

Solution

This is the case in which you can't proceed with the conversion because there are some statements (in bold) existing between the `else` and `if` keywords.

21.10 Converting a Case Decision Structure to Nested Decision Control Structures, and Vice Versa

Converting a case decision structure to nested decision control structures is quite easy. All you need to do is get the variable or the expression that the switch statement evaluates and use it in all nested decision control structures.

Case Decision Structure

```
switch (var_expr) {  
    case value-1:  
        block of statements 1  
        break;  
    case value-2:  
        block of statements 2  
        break;  
    case value-3:  
        block of statements 3  
        break;  
    .  
    .  
    .  
    case value-N:  
        block of statements N  
        break;  
    default:  
        block of statements N+1  
}
```



Nested Decision Control Structures

```
if (var_expr == value-1) {  
    block of statements 1  
}  
else {  
    if (var_expr == value-2) {  
        block of statements 2  
    }  
    else {  
        if (var_expr == value-3) {  
            block of statements 3  
        }  
        .  
        .  
        .  
        else {  
            if (var_expr == value-N) {  
                block of statements N  
            }  
            else {
```

```
        block of statements N+1
    }
}
}
}
```

Notice: The opposite conversion (from nested decision control structures to a case decision structure) is possible as long as the `else` keyword and the succeeding `if` statement (in the nested decision control structures) are side by side without any statements existing between them, and as long as the same variable or expression is evaluated for all paths. Moreover, this variable or expression should be tested in each path to determine if it is equal to constant values `value-1`, `value-2`, and `value-N` respectively.

Exercise 21.10-1 Converting the PHP Script

Rewrite the PHP script using nested decision control structures.

```
<?php
echo "What is your name? ";
$name = trim(fgets(STDIN));

switch ($name) {
    case "John":
        echo "You are my cousin!";
        break;
    case "Aphrodite":
        echo "You are my sister!";
        break;
    case "Loukia":
        echo "You are my mom!";
        break;
    default:
        echo "Sorry, I don't know you.";
}
?>
```

Solution

All nested decision control structures should evaluate the same variable `$name`.

```
<?php
echo "What is your name? ";
$name = trim(fgets(STDIN));
```

```
if ($name == "John") {
    echo "You are my cousin!";
}
else {
    if ($name == "Aphrodite") {
        echo "You are my sister!";
    }
    else {
        if ($name == "Loukia") {
            echo "You are my mom!";
        }
        else {
            echo "Sorry, I don't know you.";
        }
    }
}
?>
```

Exercise 21.10-2 *Converting the PHP Script*

Rewrite the PHP script using the case decision structure.

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));
echo "Enter a second number: ";
$y = trim(fgets(STDIN));

if ($x == 1) {
    echo "Hello Zeus!";
}
else {
    $x = $x + $y;
    echo "Hello Olympians!";
    if ($x == 2) {
        echo "Hello to everyone!";
    }
    else {
        $x = $x - $y;
        echo "Hello again!";
    }
}
```

```
if ($x == 3) {  
    echo "Hello and goodbye!";  
}  
else {  
    echo "Goooooooood morning NY!";  
}  
}  
?  
?>
```

Solution

Once again, this is a case in which you can't proceed with the conversion because there are some statements (in bold) existing between `else` and `if` keywords.

21.11 What is Code Indentation and Why is it so Important?

As you've been reading through this book, you may wonder why space characters appear in front of the PHP statements and why these statements are not written at the leftmost edge of the paragraph, as in the following example.

```
<?php  
echo "Enter a number: ";  
$x = trim(fgets(STDIN));  
echo "Enter a second number: ";  
$y = trim(fgets(STDIN));  
if ($x > 5) {  
    echo "Variable x is greater than 5";  
}  
else {  
    $x = $x + $y;  
    echo "Hello Zeus!";  
    if ($x == 3) {  
        echo "Variable x equals to 3";  
    }  
    else {  
        $x = $x - $y;  
        echo "Hello Olympians!";  
        if ($x + $y == 24) {  
            echo "The sum of x + y equals to 24";  
        }  
        else {  
            echo "Nothing of the above";  
        }  
    }  
}
```

```
}  
}  
}  
?>
```

The answer is obvious! A code without indentation is difficult to read and understand. Anyone who reads a code written this way gets confused about the `if - else` pairing (that is, to which `if` an `else` belongs). Moreover, if a long PHP script is written this way, it is almost impossible to find, for example, the location of a forgotten closing brace `}`.

Code indentation can be defined as a way to organize your source code. Indentation formats the code using spaces or tabs in order to improve readability. Well indented code is very helpful, even if it takes some extra effort, because in the long run it saves you a lot of time when you revisit your code. Unfortunately, it is sometimes overlooked and the trouble occurs at a later time. Following a particular programming style helps you to avoid syntax and logic errors. It also helps programmers to more easily study and understand code written by others.

Notice: Code indentation is similar to the way authors visually arrange the text of a book. Instead of writing long series of sentences, they break the text into chapters and paragraphs. This action doesn't change the meaning of the text but it makes it easier to read.

All statements that appear inside a set of braces `{ }` must always be indented. For example, by indenting the statements inside a dual-alternative decision structure, you visually set them apart. As a result, anyone can tell at a glance which statements are executed when the Boolean expression evaluates to `true`, and which are executed when the Boolean expression evaluates to `false`.

Notice: Only humans have difficulty reading and understanding a script without indentation. A computer can execute any code, written with or without indentation, as long as it contains no syntax errors.

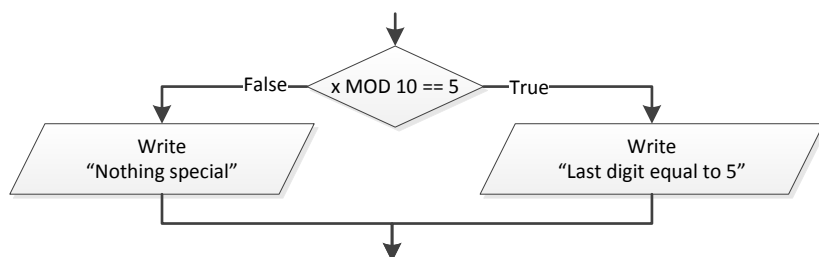
21.12 Using the “From Inner to Outer” Method in Decision Control Structures

“From inner to outer” is a method proposed by this book to help you learn “Algorithmic Thinking” from the inside out. This method first manipulates and designs the inner (nested) control structures. Then, as the algorithm is developed, more and more control structures are added, nesting the previous ones.

Let's try the following example.

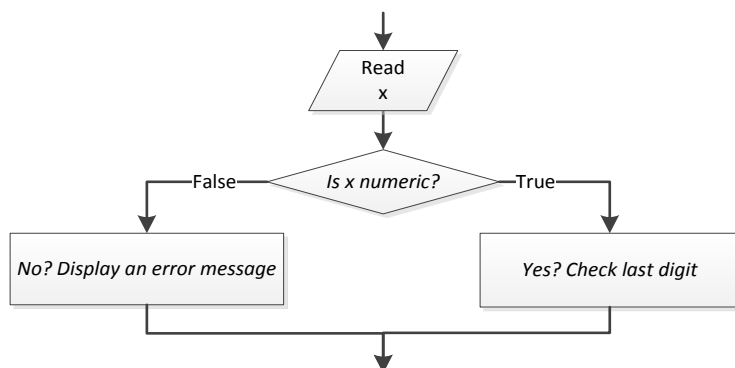
Design a flowchart that lets the user enter a number and, if its last digit is equal to 5, a message “Last digit equal to 5” is displayed; otherwise, a message “Nothing special” is displayed. Moreover, if the user enters a non-numeric character, an error message should be displayed.

First, you can design the inner decision structure that displays the messages “Last digit equal to 5” or “Nothing special” (depending on the last digit). The flowchart fragment is shown here.

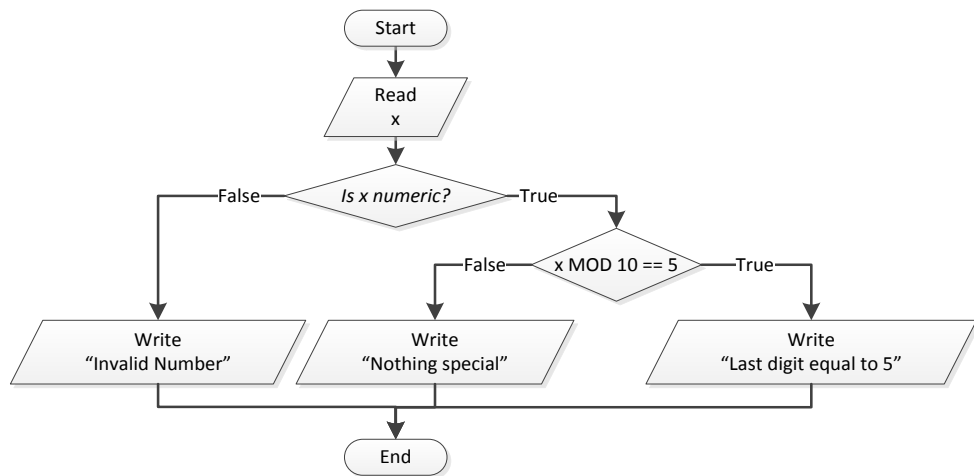


Remember! You can isolate the last digit of any number using the modulus 10 operation. It is not necessary to know the exact number of digits.

Now let's deal with the data validation problem. In this exercise, you need to test whether or not the user has entered a valid numeric value. If not, an error message should be displayed; otherwise, you can proceed to check the last digit. The outer decision structure is shown in the flowchart that follows.



Now, you can combine both flowchart fragments, nesting the first into the second one and the final flowchart becomes



Now you may wonder why we go to all this trouble. It was too easy and it could have been designed as usual without the use of the “from inner to outer” method. Yes, you are probably right. But imagine this method used in very large and complicated control structures. It can help you design error-free flowcharts or even PHP scripts and save you a lot of effort! This book uses this method wherever and whenever it seems necessary.

21.13 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Smaller algorithms are always the best solution to a given problem.
2. You can move a statement outside, and right before, a dual-alternative decision structure as long as it exists at the beginning of both paths of the decision structure.
3. You can always use a logical operator instead of nested decision control structures to increase readability.
4. Two single-alternative decision structures can be merged into one single-alternative decision only when they are in a row and when they evaluate the same Boolean expression.
5. Conversion from a dual-alternative decision structure to two single-alternative decision structures is always possible.
6. Two single-alternative decision structures can be “merged” into (actually, replaced by) one dual-alternative decision only when they are in a row and only when they evaluate the same Boolean expression.
7. Conversion from a case decision structure to a multiple-alternative decision structure is always possible.
8. Conversion from a multiple-alternative decision structure to a case decision structure is always possible.
9. Conversion from a multiple-alternative decision structure to nested decision control structures is always possible.

10. Conversion from nested decision control structures to a multiple-alternative decision structure is always possible.
11. Conversion from a case decision structure to nested decision control structures is always possible.
12. Conversion from nested decision control structures to a case decision structure is always possible.
13. PHP scripts written without code indentation cannot be executed by a computer.

21.14 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The following two scripts

```
<?php
$a = trim(fgets(STDIN));
if ($a > 40) {
    $a++;
    echo $a * 2;
}
else {
    $a++;
    echo $a * 3;
}
?>
```

```
<?php
$a = trim(fgets(STDIN));
$a++;
if ($a > 40) {
    echo $a * 2;
}
else {
    echo $a * 3;
}
?>
```

- a. produce the same result.
 - b. produce the same result, but the right script is faster.
 - c. do not produce the same result.
 - d. none of the above
2. The following two scripts

```
<?php
$a = trim(fgets(STDIN));
if ($a > 40) {
    echo $a * 2;
}
if ($a > 40) {
    echo $a * 3;
}
?>
```

```
<?php
$a = trim(fgets(STDIN));
if ($a > 40) {
    echo $a * 2;
    echo $a * 3;
}
?>
```

- a. produce the same results, but the left script is faster.
- b. produce the same results, but the right script is faster.

c. do not produce the same results.

d. none of the above

3. The following two scripts

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$y = 5;

if ($a > 15 || $b < 25)
    $y--;
else
    $y++;

echo $y;
?>
```

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$y = 5;

if ($a > 15)
    $y--;
elseif ($b < 25)
    $y--;
else
    $y++;
echo $y;
?>
```

a. produce the same result.

b. do not produce the same result.

c. neither of the above

4. The following two scripts

```
<?php
$a = trim(fgets(STDIN));
if ($a > 40) {
    echo $a * 2;
}
else {
    echo $a * 3;
}
?>
```

```
<?php
$a = trim(fgets(STDIN));
if ($a > 40) {
    echo $a * 2;
}
if ($a <= 40) {
    echo $a * 3;
}
?>
```

a. produce the same result(s), but the left script is faster.

b. produce the same result(s), but the right script is faster.

c. do not produce the same result(s).

d. none of the above

5. The following code fragment

```
if ($a == 5)
    echo "Hello Zeus!";
elseif ($b == 6)
```

```
echo "Hello Hermes!";  
elseif ($c == 7)  
echo "It's all Greek to me!";
```

- a. cannot be converted to a case decision structure because it does not contain the `else` keyword.
- b. cannot be converted to a case decision structure because it does not evaluate the same variable for all paths.
- c. neither of the above

6. The following code fragment

```
if ($a == 1) {  
    echo "Hello Dionysus!";  
}  
else {  
    $a = $a * $x;  
    if ($a == 5) {  
        echo "Hello Hera!";  
    }  
    else {  
        echo "It's all Greek to me!";  
    }  
}
```

- a. cannot be converted to a multiple-alternative decision structure because it evaluates the same variable for all paths.
- b. can be converted to a multiple-alternative decision structure.
- c. cannot be converted to a multiple-alternative decision structure because of the `$a = $a * $x` statement.
- d. none of the above

7. The following code fragment

```
if ($a == 1) {  
    echo "Hello Poseidon!";  
}  
else {  
    $x = $a * $x;  
    if ($x == 5) {  
        echo "Hello Hephaestus!";  
    }  
    else {  
        echo "It's all Greek to me!";  
    }  
}
```

```
}
```

- a. cannot be converted to a case decision structure because of the $\$x = \$a * \$x$ statement.
- b. cannot be converted to a case decision structure because it does not evaluate the same variable for all paths.
- c. both of the above

8. The following script

```
<?php
$x = trim(fgets(STDIN));
if ($x < 0) {
    $x = (-1) * $x;
    echo $x;}
?>
```

cannot be executed by a computer because

- a. it does not use code indentation.
- b. it includes syntax errors.
- c. it includes logic errors.
- d. none of the above

21.15 Review Exercises

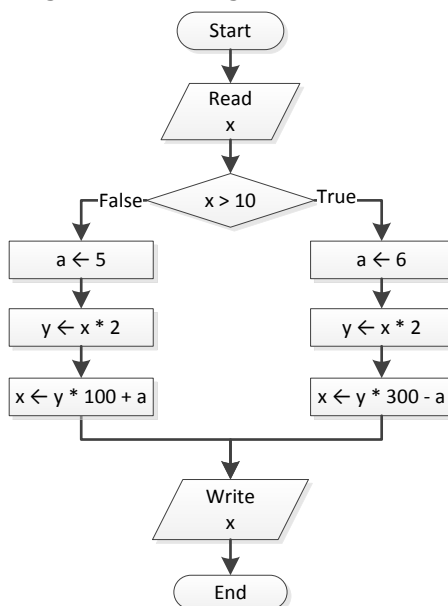
Complete the following exercises.

1. Rewrite the following PHP script using fewer statements.

```
<?php
$y = trim(fgets(STDIN));

if ($a > 0) {
    $x = trim(fgets(STDIN));
    $a = $x * 4 * $y;
    echo $y;
    $a++;
}
else {
    $x = trim(fgets(STDIN));
    $a = $x * 2 * $y + 7;
    echo $y;
    $a--;
}
echo $a;
?>
```

2. Redesign the following flowchart using fewer statements.



3. Rewrite the following PHP script using fewer statements.

```
<?php
$a = trim(fgets(STDIN));

if ($a < 1) {
    $y = 5 + $a;
    echo $y;
}
elseif ($a < 5) {
    $y = 23 / $a;
    echo $y;
}
elseif ($a < 10) {
    $y = 5 * $a;
    echo $y;
}
else {
    echo "Error!";
}

?>
```

4. Rewrite the following PHP script using logical operators.

```
<?php
```

```
$day = trim(fgets(STDIN));
$month = trim(fgets(STDIN));
$name = trim(fgets(STDIN));

if ($day == 16) {
    if ($month == 2) {
        if ($name == "Loukia") {
            echo "Happy Birthday!!!";
        }
        else {
            echo "No match!";
        }
    }
    else {
        echo "No match!";
    }
}
else {
    echo "No match!";
}
?>
```

5. A teacher asks her students to rewrite the following PHP script without using logical operators.

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));

if ($a > 10 && $c < 2000) {
    $d = ($a + $b + $c) / 12;
    echo "The result is: ", $d;
}
else {
    echo "Error!";
}
?>
```

One student wrote the following PHP script:

```
<?php
$a = trim(fgets(STDIN));
```

```
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));

if ($a > 10) {
    if ($c < 2000) {
        $d = ($a + $b + $c) / 12;
        echo "The result is: ", $d;
    }
    else {
        echo "Error!";
    }
}
?>
```

Determine if the script produces the same results for all possible paths as the one given by his teacher. If not, try to modify it and make it work the same way.

6. Rewrite the following PHP script using only single-alternative decision structures.

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));

if ($a > 10) {
    if ($b < 2000) {
        if ($c != 10) {
            $d = ($a + $b + $c)/12;
            echo "The result is: ", $d;
        }
    }
}
else {
    echo "Error!";
}
?>
```

7. In the following PHP script, “merge” the single-alternative decision structures.

```
<?php
$a = trim(fgets(STDIN));
```

```
$y = 3;
if ($a > 0) {
    $y = $y * $a;
}
$b = trim(fgets(STDIN));
if (!$a <= 0) {
    echo "Hello Zeus";
}

echo $y, $b;
?>
```

8. In the following PHP script, “merge” the single-alternative decision structures.

```
<?php
$a = trim(fgets(STDIN));

$y = 0;
if ($a > 0) {
    $y = $y + 7;
}
$b = trim(fgets(STDIN));
if (!$a > 0) {
    echo "Hello Zeus";
}
if ($a <= 0) {
    echo abs($a);
}
echo $y;
?>
```

9. According to research from 2013, the most popular operating system on tablet computers was iOS, with Android being in second place and Microsoft Windows in last place. In the following PHP script, rearrange the Boolean expressions to make the script run more efficiently for most of the cases.

```
<?php
echo "What is your tablet's OS? ";
$os = trim(fgets(STDIN));

if ($os == "Windows") {
    echo "Microsoft";
}
```

```
elseif ($os == "iOS" ) {  
    echo "Apple";  
}  
elseif ($os == "Android") {  
    echo "Google";  
}  
?  
>
```

10. Rewrite the following PHP script using the multiple-alternative decision structure.

```
<?php  
$a = trim(fgets(STDIN));  
$x = trim(fgets(STDIN));  
$y = trim(fgets(STDIN));  
  
switch ($a) {  
    case 3:  
        $x = $x / 4;  
        $y = pow($y, 5);  
        break;  
    case 7:  
        $x = $x * 3;  
        $y++;  
        break;  
    case 22:  
        $x = $x % 4;  
        $y += 9;  
        break;  
    default:  
        $x -= 9;  
        $y++;  
}  
  
echo $x, $y;  
?  
>
```

11. Rewrite the PHP script of the previous exercise using nested decision control structures.
12. Rewrite the following PHP script using the case decision structure.

```
<?php  
echo "Enter a color in hexadecimal: ";
```



```
$color = trim(fgets(STDIN));

if ($color == "FF0000") {
    echo "red";
}
elseif ($color == "00FF00") {
    echo "green";
}
elseif ($color == "0000FF") {
    echo "blue";
}
elseif ($color == "FFFFFF") {
    echo "white";
}
elseif ($color == "000000") {
    echo "black";
}
elseif ($color == "7F7F7F") {
    echo "gray";
}
else {
    echo "Unknown color!";
}

?>
```

13. Rewrite the PHP script of the previous exercise using nested decision control structures.
14. Write the following PHP script using correct indentation and then rewrite it using a multiple-alternative decision structure.

```
<?php
$a = trim(fgets(STDIN));

if ($a > 1000)
    echo "Big Positive";
else {
    if ($a > 0)
        echo "Positive";
    else {
        if ($a < -1000)
            echo "Big Negative";
        else {
```

```
if ($a < 0)
echo "Negative";
else
echo "Zero";
}
}
}
?>
```

15. Write the following PHP script using correct indentation and then rewrite it using nested decision control structures.

```
<?php
$a = trim(fgets(STDIN));

if ($a < 1) {
    $y = 5 + $a;
    echo $y;
}
elseif ($a < 5) {
    $y = 23 / $a;
    echo $y;
}
elseif ($a < 10) {
    $y = 5 * $a;
    echo $y;
}
else {
    echo "Error!";
}
?>
```

Chapter 22

Flowcharts with Decision Control Structures

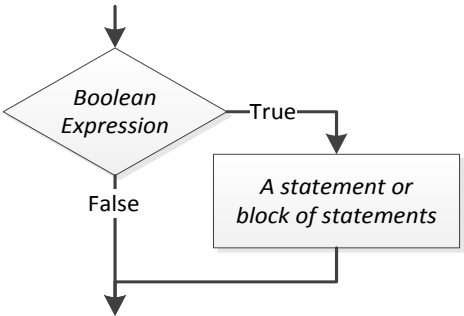
22.1 Introduction

By working through the previous chapters, you’ve become familiar with all the decision control structures. Since flowcharts are an ideal way to learn “Algorithmic Thinking” and to help you better understand specific control structures, this chapter is dedicated to teaching you how to convert a PHP script to a flowchart, or a flowchart to a PHP script.

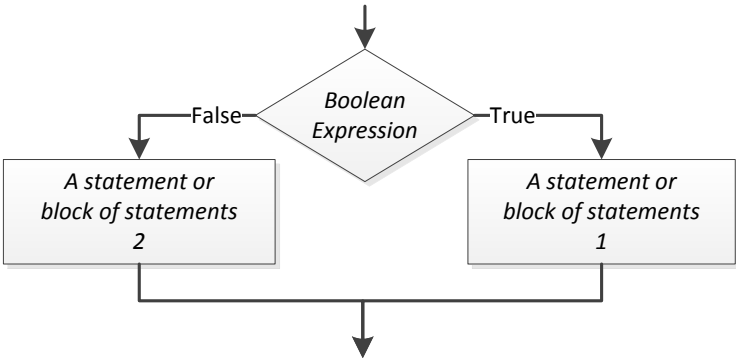
22.2 Converting PHP Scripts to Flowcharts

To convert a PHP script to its corresponding flowchart, you need to recall all the decision control structures and their corresponding flowcharts. They are all summarized here.

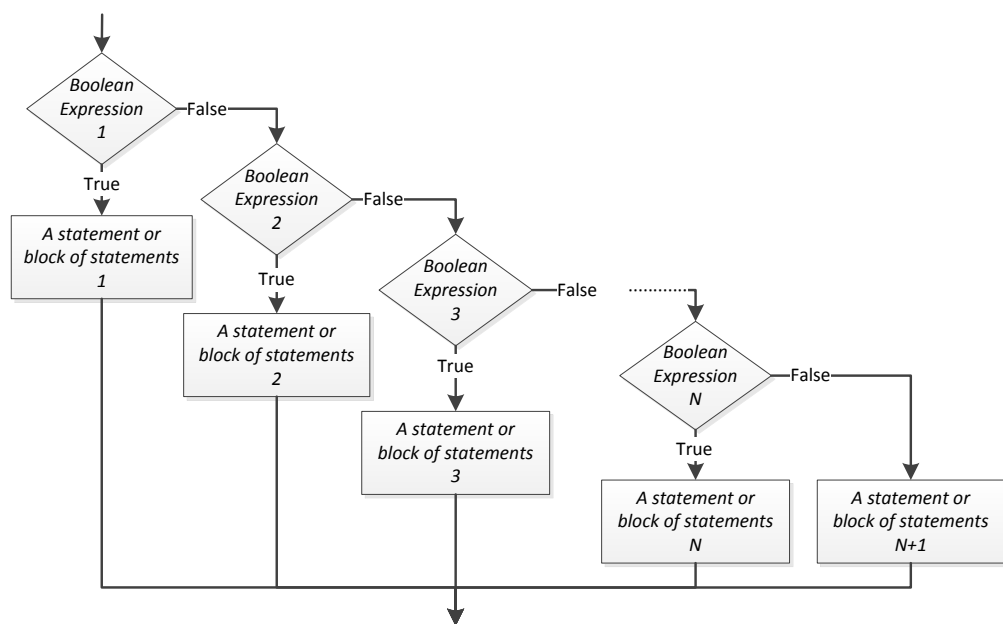
The single-alternative decision structure



The dual-alternative decision structure



The multiple-alternative decision structure and the case decision structure



Exercise 22.2-1 Designing the Flowchart

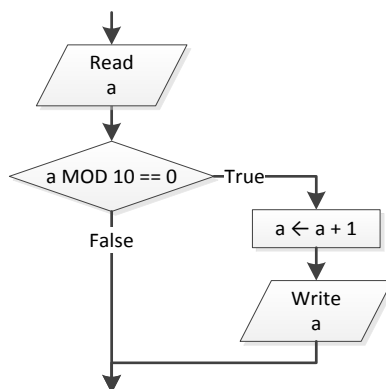
Design the flowchart that corresponds to the following code fragment.

```

$a = trim(fgets(STDIN));
if ($a % 10 == 0) {
    $a++;
    echo $a;
}
  
```

Solution

This is quite easy! It uses a single decision control structure. The corresponding flowchart is as follows.



Exercise 22.2-2 *Designing the Flowchart*

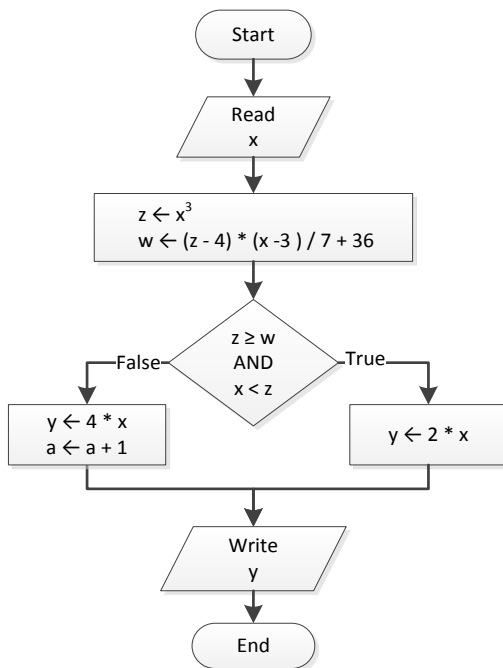
Design the flowchart that corresponds to the following PHP script.

```
<?php
    $x = trim(fgets(STDIN));

    $z = pow ($x, 3);
    $w = ($z - 4) * ($x - 3) / 7 + 36;
    if ($z >= $w && $x < $z) {
        $y = 2 * $x;
    }
    else {
        $y = 4 * $x;
        $a++;
    }
    echo $y;
?>
```

Solution

This PHP script uses a dual-alternative decision structure. Its corresponding flowchart is as follows.



Remember! A flowchart is a very loose method of representing an algorithm. Thus, it is quite permissible to write x^3 or even to use the PHP function `pow()`. Do whatever you wish; everything is permitted, on condition that anyone familiar with flowcharts can clearly understand what you are trying to say!

Exercise 22.2-3 Designing the Flowchart

Design the flowchart that corresponds to the following code fragment given in general form.

```

if (Boolean_Expression A) {
    Block of statements A1
    if (Boolean_Expression B) {
        Block of statements B1
    }
    Block of statements A2
}
else {
    Block of statements A3
    if (Boolean_Expression C) {
        Block of statements C1
    }
    else {

```

```
    Block of statements C2
  }
}
```

Solution

For better observation, the initial code fragment is presented again with all the nested decision control structures enclosed in rectangles.

```
if (Boolean_Expression A) {
    Block of statements A1
```

This is a nested
decision control structure

```
    if ( Boolean_Expression B ) {
        Block of statements B1
    }
```

```
    Block of statements A2
}
```

```
else {
```

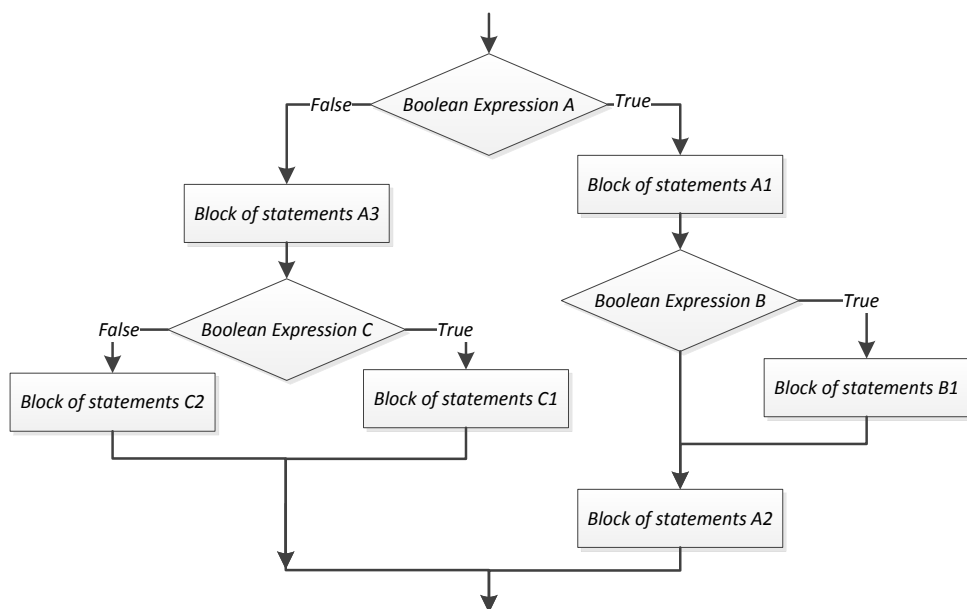
```
    Block of statements A3
```

This is a nested
decision control structure

```
    if ( Boolean_Expression C ) {
        Block of statements C1
    }
    else {
        Block of statements C2
    }
```

```
}
```


and the flowchart fragment in general form is shown here.



Exercise 22.2-4 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

```

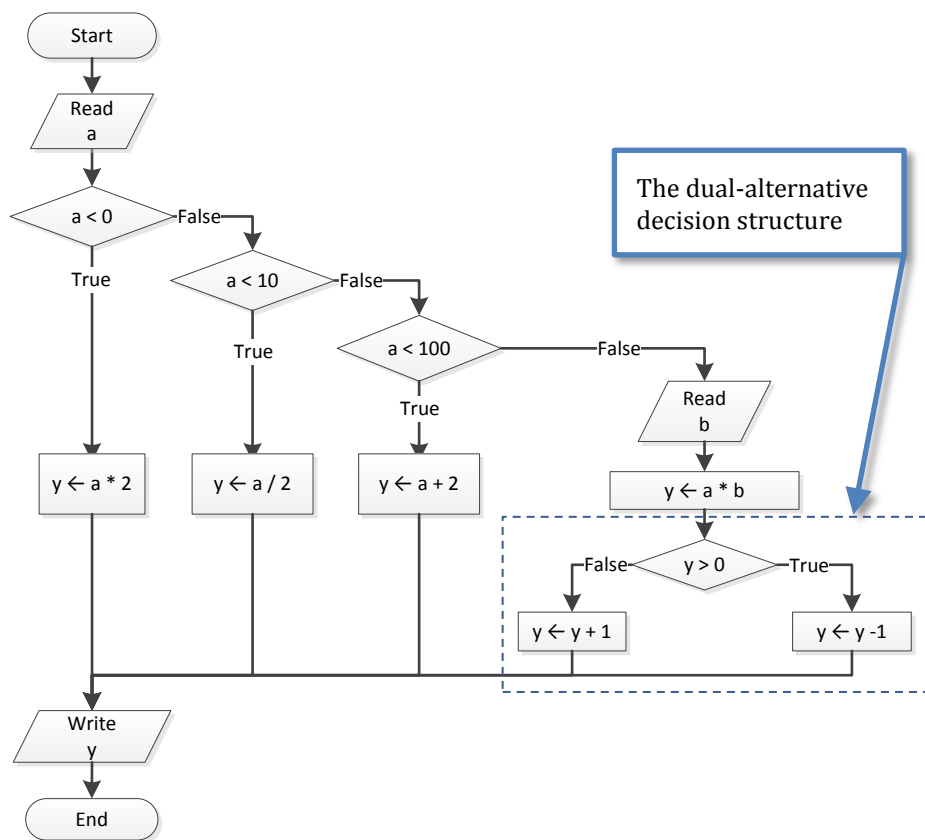
<?php
$a = trim(fgets(STDIN));
if ($a < 0)
    $y = $a * 2;
elseif ($a < 10)
    $y = $a / 2;
elseif ($a < 100)
    $y = $a + 2;
else {
    $b = trim(fgets(STDIN));
    $y = $a * $b;
    if ($y > 0)
        $y--;
    else
        $y++;
}
echo $y;
  
```

?>

Solution

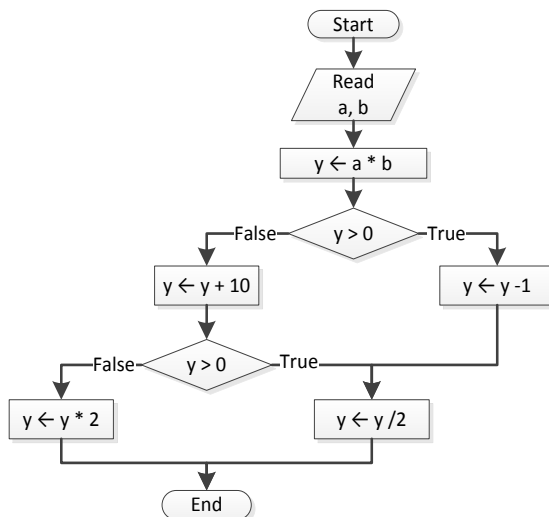
In this PHP script, a dual-alternative decision structure (`if ($y > 0)`) is nested within a multiple-alternative decision structure.

The flowchart is shown here.



22.3 Converting Flowcharts to PHP Scripts

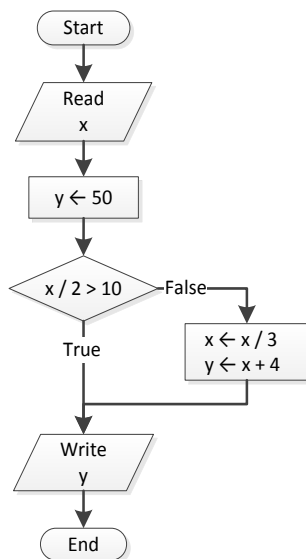
This conversion is not always an easy one. There are cases in which the flowchart designers follow no particular rules, so the initial flowchart may need some modifications before it can become a PHP script. An example of one such case is as follows.



As you can see, the decision control structures included in this flowchart fragment match none of the structures that you have already learned, such as the single-alternative, the dual-alternative, and the multiple-alternative. Thus, you have only one choice and this is to modify the flowchart by adding extra statements or removing existing ones until known decision control structures start to appear. Following are some exercises in which the initial flowchart does need modification.

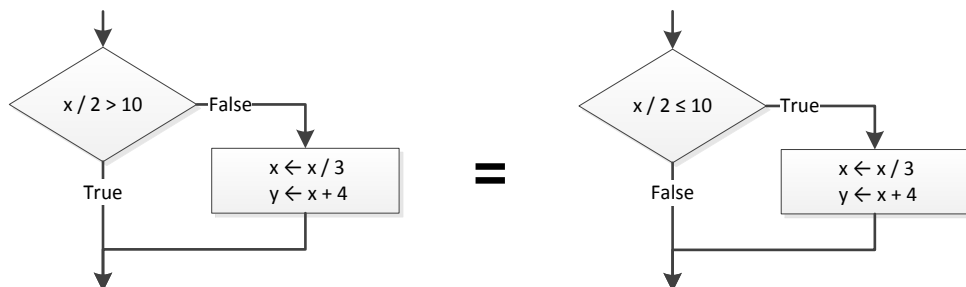
Exercise 22.3-1 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart.

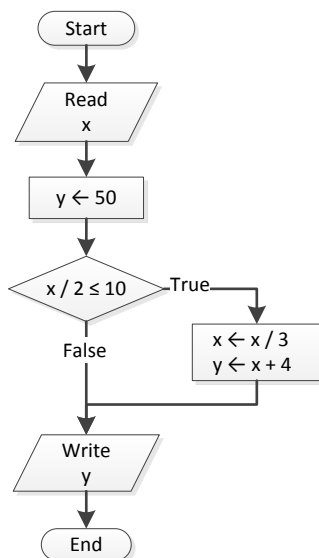


This is quite easy. The only obstacle you must overcome is that the true and false paths are not quite in the right positions. You need to use the true path, and not the false path, to actually include the statements in the decision control structure.

It is possible to switch the two paths, but you also need to negate the corresponding Boolean expression. The following two flowchart fragments are equivalent.



Thus, the flowchart now looks like this.



and the corresponding PHP script is shown here.

```
<?php
```

```
$x = trim(fgets(STDIN));
```

```
$y = 50;
```

```
if ($x / 2 <= 10) {
```

```
    $x = $x / 3;
```

```
    $y = $x + 4;
```

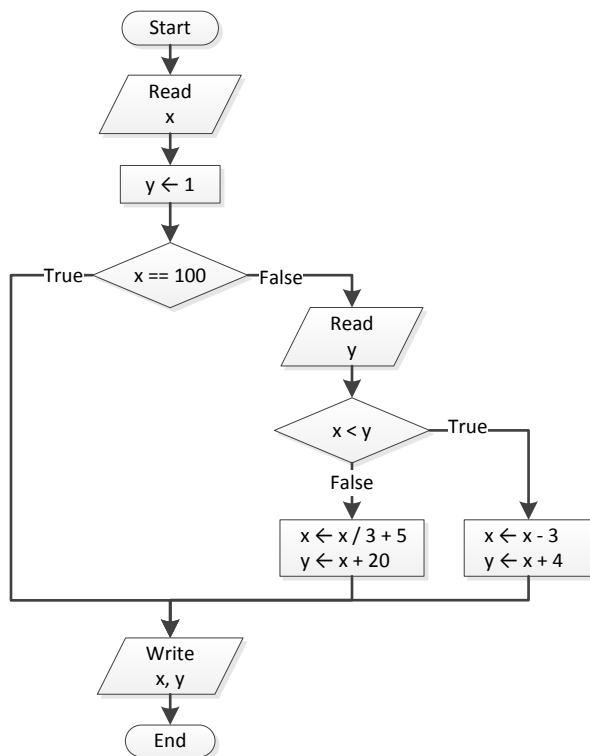
```

}
echo $y;
?>

```

Exercise 22.3-2 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart.



Solution

Once again, you need to negate the Boolean expression `x == 100` and switch the true/false paths. The PHP script is shown here.

```

<?php
    $x = trim(fgets(STDIN));

    $y = 1;

    if ($x != 100) {
        $y = trim(fgets(STDIN));
        if ($x < $y) {
            $x = $x - 3;

```

```

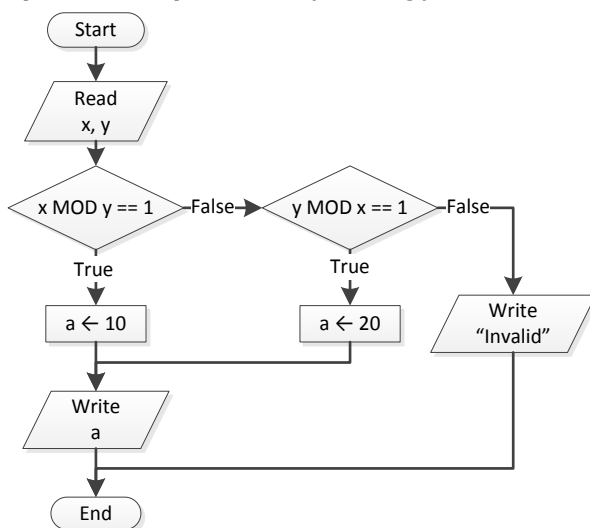
    $y = $x + 4;
}
else {
    $x = $x / 3 + 5;
    $y = $x + 20;
}

echo $x, $y;
?>

```

Exercise 22.3-3 Writing the PHP Script

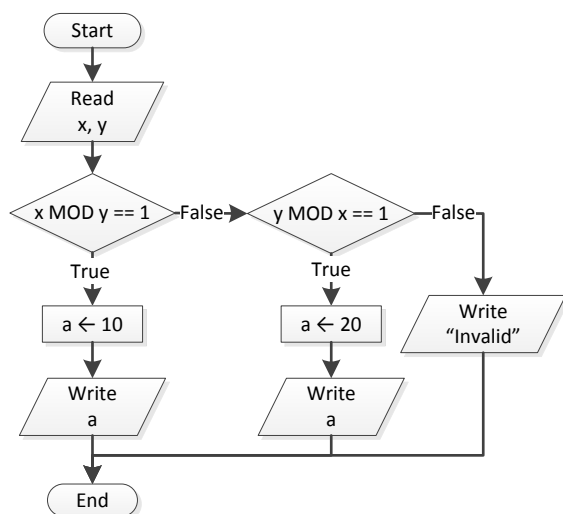
Write the PHP script that corresponds to the following flowchart.



Solution

In this flowchart, the decision control structures match none of the decision control structures that you have already learned, such as the single-alternative, the dual-alternative, and the multiple-alternative. Thus, you must modify the flowchart by adding extra statements or removing existing ones until known decision control structures start to appear!

The obstacle you must overcome in this exercise is the decision control structure that evaluates the `y MOD x == 1` Boolean expression. Note that when flow of execution follows the false path it executes the statement `a <- 20` and then the statement `Write a`. Thus, if you simply add a new statement, `Write a`, inside its true path you can keep the flow of execution intact. The following flowchart is equivalent to the initial one.



and the PHP script is

```
<?php
```

```
$x = trim(fgets(STDIN));
```

```
$y = trim(fgets(STDIN));
```

```
if ($x % y == 1) {
```

```
    $a = 10;
```

```
    echo $a;
```

```
}
```

```
elseif ($y % $x == 1) {
```

```
    $a = 20;
```

```
    echo $a;
```

```
}
```

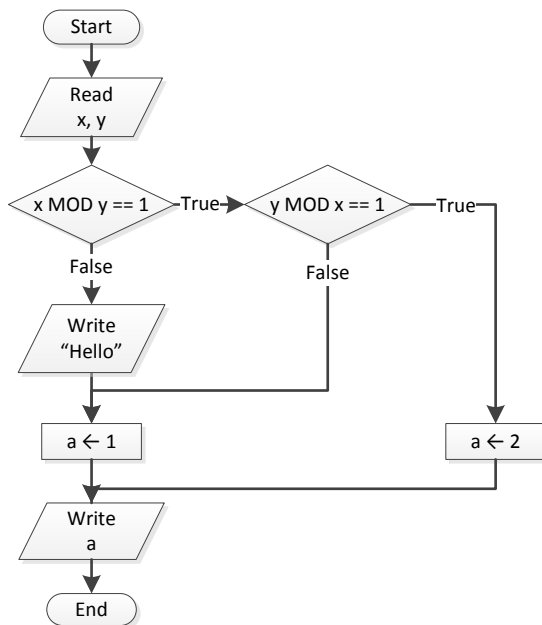
```
else
```

```
    echo "Invalid";
```

```
?>
```

Exercise 22.3-4 Writing the PHP Script

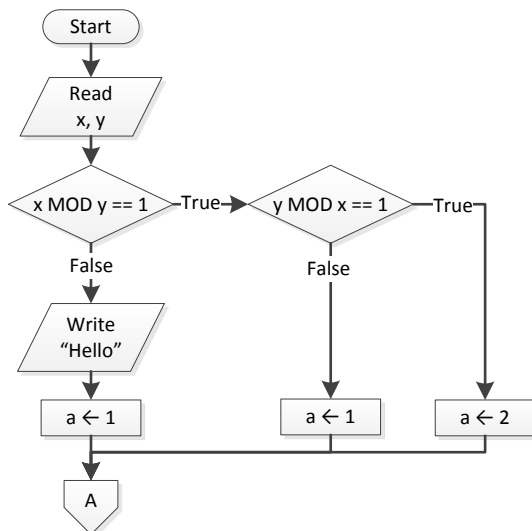
Write the PHP script that corresponds to the following flowchart.

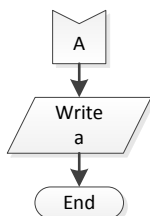


Solution

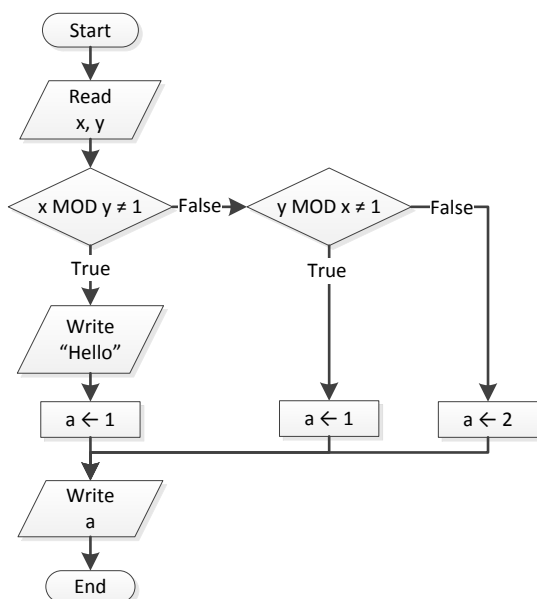
In this flowchart, the decision control structures match none of the already known decision control structures, so you must modify the flowchart!

The obstacle that you must overcome is the decision control structure that evaluates the $y \text{ MOD } x == 1$ Boolean expression. Note that when the flow of execution follows its false path, it executes the statement $a \leftarrow 1$. Thus, if you simply add an extra statement $a \leftarrow 1$ inside its false path, you can keep the flow of execution intact. The following flowchart is equivalent to the initial one.





Now the flowchart contains known decision control structures and it can become a PHP script. But don't be in a hurry! This flowchart, as is, can be written in PHP only by using nested `if` statements. However, there is something better that you can do! If you negate all Boolean expressions and also switch their true/false paths, you can have a multiple-alternative decision structure, which is more convenient in PHP than nested control structures. The modified flowchart is shown here.



and the PHP script is

```
<?php
```

```
$x = trim(fgets(STDIN));
```

```
$y = trim(fgets(STDIN));
```

```
if ($x % y != 1) {
```

```
    echo "Hello";
```

```
    $a = 1;
```

```
}
```

```
elseif ($y % $x != 1)
```

```
    $a = 1;
```

```
else
```

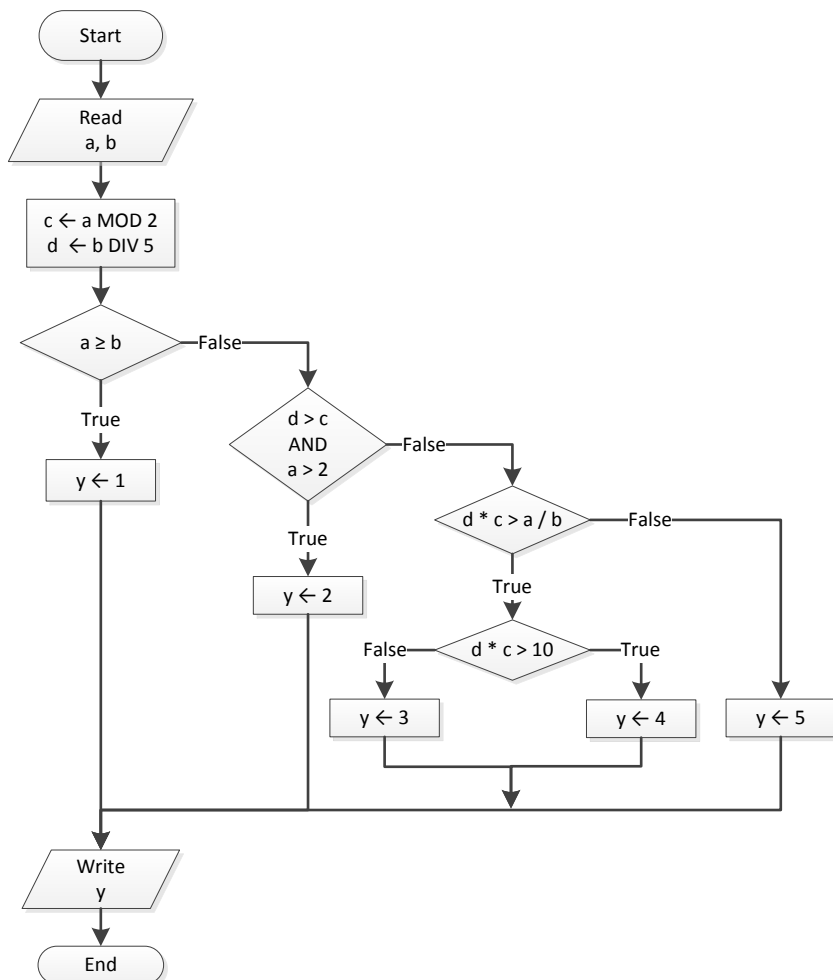
```
$a = 2;

echo $a;

?>
```

Exercise 22.3-5 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart.



Solution

Please take a closer look at the statement $d \leftarrow b \text{ DIV } 5$. The operator `DIV` is widely used in flowcharts to calculate the quotient of an integer division. Unfortunately, PHP does not incorporate such an operator but you can still use the `intval()` function to achieve the same result.

This exercise uses a multiple-alternative decision structure and a dual-alternative decision structure nested inside the first one.

The PHP script is shown here.

```
<?php
```

```
    $a = trim(fgets(STDIN));
```

```
    $b = trim(fgets(STDIN));
```

```
    $c = $a % 2;
```

```
    $d = intval($b / 5);
```

```
    if ($a >= $b)
```

```
        $y = 1;
```

```
    elseif ($d > $c && $a > 2)
```

```
        $y = 2;
```

```
    elseif ($d * $c > a / b) {
```

```
        if ($d * $c > 10)
```

```
            $y = 4;
```

```
        else
```

```
            $y = 3;
```

```
    }
```

```
    else
```

```
        $y = 5;
```

```
    echo $y;
```

```
?>
```

This is the dual-alternative decision structure

22.4 Review Exercises

Complete the following exercises.

1. Design the flowchart that corresponds to the following PHP script.

```
<?php
```

```
    $a = trim(fgets(STDIN));
```

```
    if ($a % 10 == 0) {
```

```
        $a++;
```

```
        echo "Message #1";
```

```
    }
```

```
    if ($a % 3 == 1) {
```

```
        $a += 5;
```

```
        echo "Message #2";
```

```
    }
```

```
if ($a % 3 == 2) {  
    $a += 10;  
    echo "Message #3";  
}  
  
echo $a;  
?>
```

2. Design the flowchart that corresponds to the following PHP script.

```
<?php  
$a = trim(fgets(STDIN));  
  
if ($a % 10 == 0) {  
    $a++;  
    echo "Message #1";  
}  
  
if ($a % 3 == 1) {  
    $a += 5;  
    echo "Message #2";  
}  
else {  
    $a += 7;  
}  
echo $a;  
?>
```

3. Design the flowchart that corresponds to the following PHP script.

```
<?php  
$a = trim(fgets(STDIN));  
  
if ($a < 0) {  
    $y = $a * 2;  
    if ($y > 0)  
        $y += 2;  
    elseif ($y == 0)  
        $y *= 6;  
    else  
        $y /= 7;  
}  
elseif ($a < 22)
```

```
$y = $a / 3;
elseif ($a < 32)
    $y = $a - 7;
else {
    $b = trim(fgets(STDIN));
    $y = $a - $b;
}
echo $y;
?>
```

4. Design the flowchart that corresponds to the following code fragment given in general form.

```
if (Boolean_Expression A) {
    if (Boolean_Expression B) {
        Block of statements B1
    }
    else {
        Block of statements B2
    }
    Block of statements A1
}
else {
    Block of statements A2
    if (Boolean_Expression C) {
        Block of statements C1
    }
    elseif (Boolean_Expression D) {
        Block of statements D1
    }
    else {
        Block of statements E1
    }
    Block of statements A3
}
```

5. Design the flowchart that corresponds to the following PHP script.

```
<?php
$a = trim(fgets(STDIN));

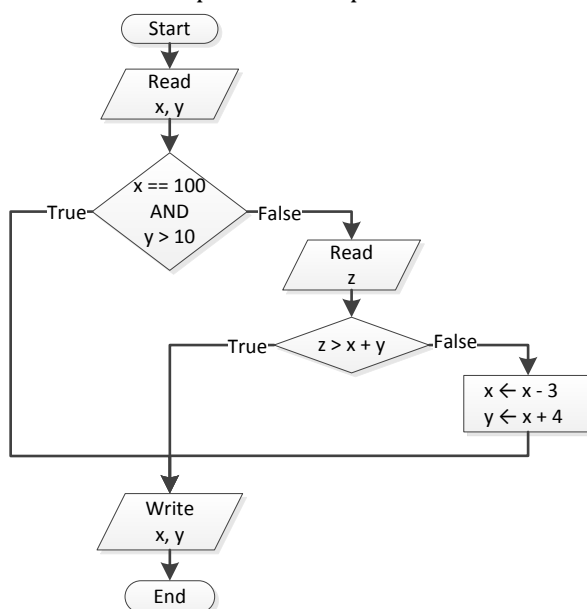
switch ($a) {
    case 1:
```

```

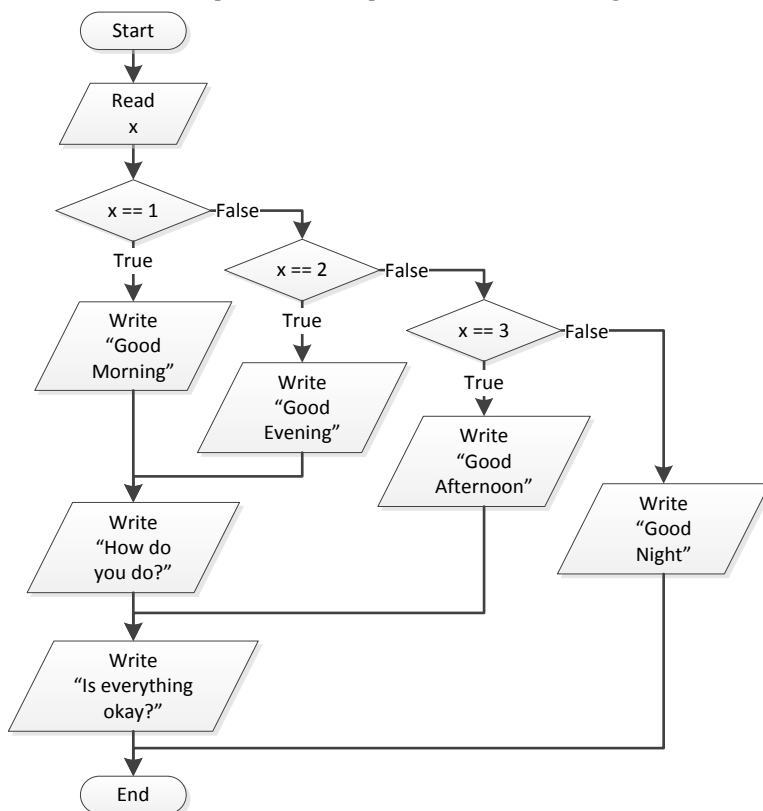
    $y = $a * 2;
    break;
case 2:
    $y = $a / 3;
    break;
case 3:
    $y = $a + 3;
    if ($y % 2 == 1)
        $y += 2;
    elseif ($y == 0)
        $y *= 6;
    else
        $y /= 7;
    break;
case 4:
    $b = trim(fgets(STDIN));
    $y = $a + $b + 2;
    break;
}
echo $y;
?>

```

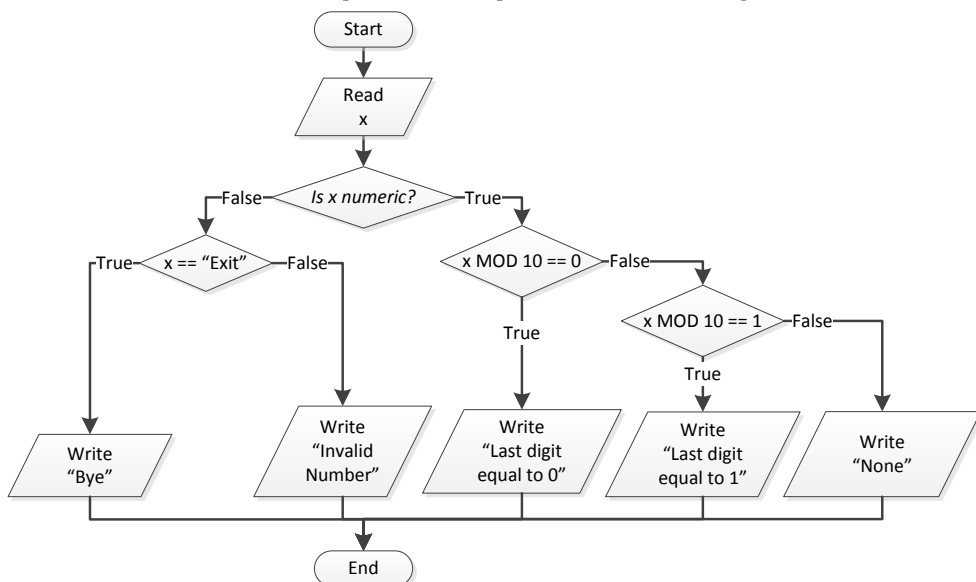
6. Write the PHP script that corresponds to the following flowchart.



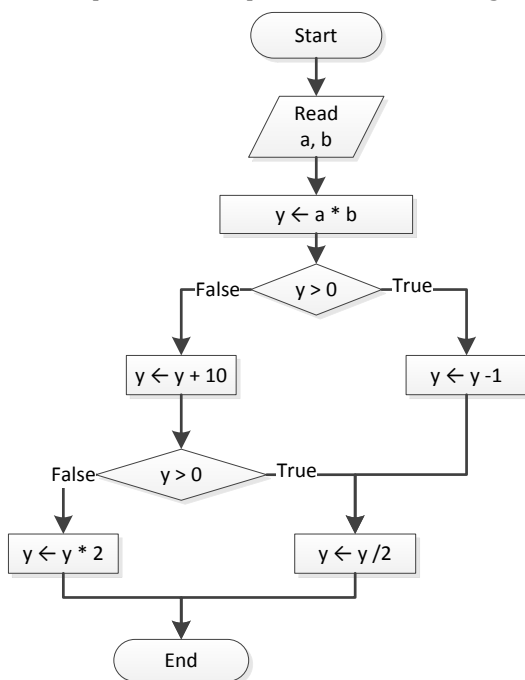
7. Write the PHP script that corresponds to the following flowchart.



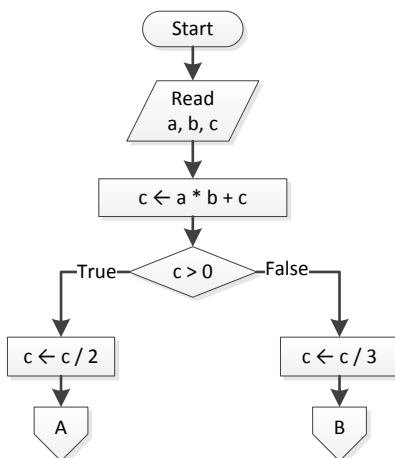
8. Write the PHP script that corresponds to the following flowchart.

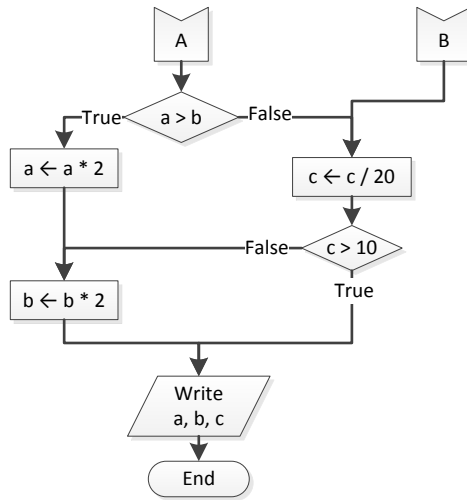


9. Write the PHP script that corresponds to the following flowchart.



10. Write the PHP script that corresponds to the following flowchart.





Chapter 23

More Exercises with Decision Control Structures

23.1 Simple Exercises with Decision Control Structures

Exercise 23.1-1 *Both Odds or Both Evens?*

Write a PHP script that prompts the user to enter two integers and then displays a message indicating whether both numbers are odd or both are even; otherwise a message “Nothing special” is displayed.

Solution

The PHP script is shown here.

file_23.1-1.php

```
<?php
echo "Enter an integer: ";
$n1 = trim(fgets(STDIN));
echo "Enter a second integer: ";
$n2 = trim(fgets(STDIN));

if ($n1 % 2 == 0 && $n2 % 2 == 0) {
    echo "Both integers are evens";
}
elseif ($n1 % 2 != 0 && $n2 % 2 != 0) {
    echo "Both integers are odds";
}
else {
    echo "Nothing special!";
}
?>
```

Exercise 23.1-2 *Validating Data Input and Finding if a Number is Exactly Divisible by both 5 and 8?*

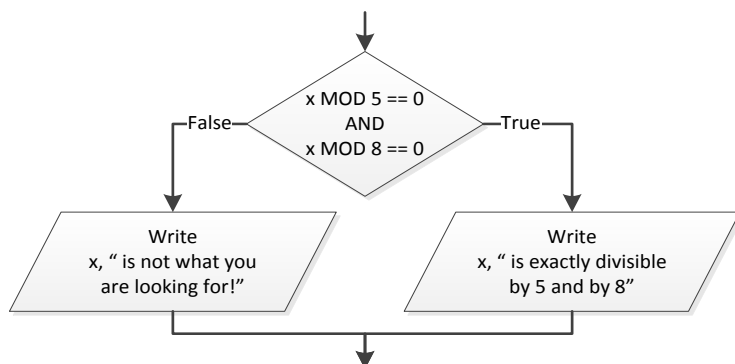
Design a flowchart and write the corresponding PHP script that prompts the user to enter an integer and then displays a message indicating whether this number is exactly divisible by 5 and by 8. For example, 40 is such a number. Moreover, if the user enters non-numeric characters, an error message should be displayed.

(This exercise gives you some practice in working with data validation.)

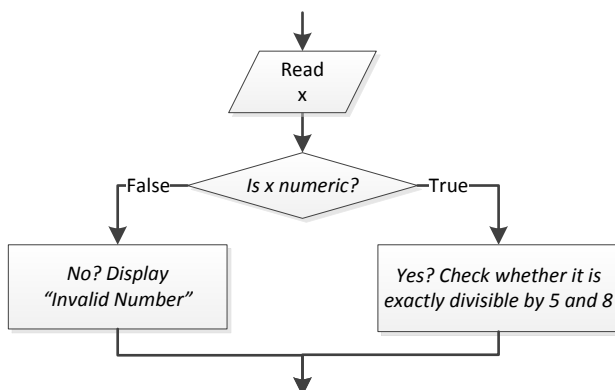
Solution

Let's design the flowchart using the “from inner to outer” method.

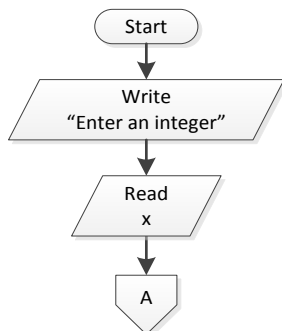
First, you need to design the inner (nested) decision control structure without taking data validation into consideration. Suppose variable x contains the value entered by the user. The inner decision control structure is shown in the flowchart that follows.

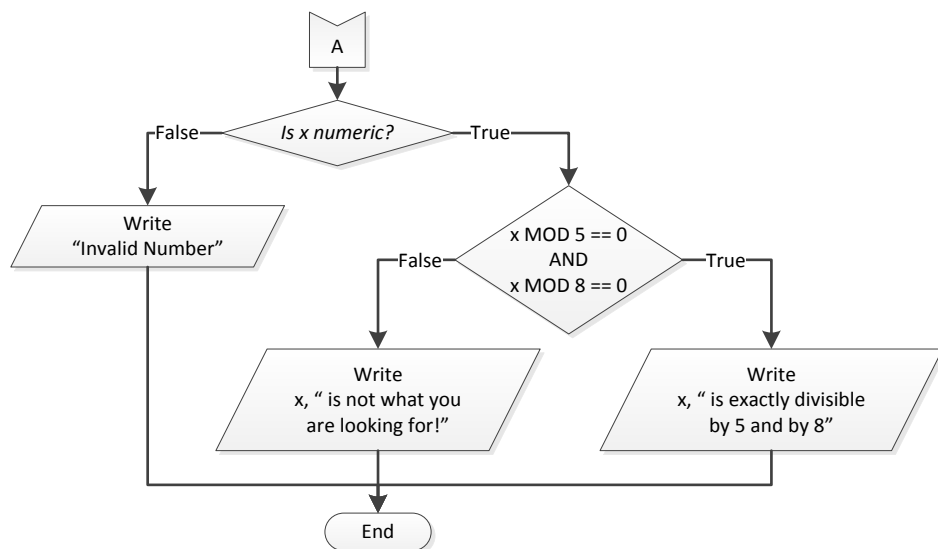


Now it's time to handle data validation. You need to test whether the user enters numeric or non-numeric characters. The flowchart fragment given in general form is shown here.



Combining both flowcharts, the final flowchart looks like this.





In PHP, you can use the `is_numeric()` function to test whether or not a variable contains a numeric value. The PHP script is shown here.

file_23.1-2.php

```

<?php
echo "Enter an integer: ";
$x = trim(fgets(STDIN));

if (is_numeric($x) == true) {
    if ($x % 5 == 0 && $x % 8 == 0) {
        echo $x, " is exactly divisible by 5 and by 8";
    }
    else {
        echo $x, " is not what you are looking for!";
    }
}
else {
    echo "Invalid Number";
}
?>

```

Exercise 23.1-3 Is it an Integer?

Write a PHP script that prompts the user to enter a number and then displays a message indicating whether the data type of this number is integer or real. Moreover, if the user enters a non-numeric character, an error message should be displayed.

(This exercise gives you some more practice in working with data validation.)

Solution

It is well known that a number is considered an integer when it contains no fractional part. In PHP, you can use the function `intval()` to return the integer value of any real number. If the number given is equal to its integer value, then the number is considered an integer.

For example, if the user enters the integer 7, this number and its integer value, `intval(7)`, are equal.

On the other hand, if the user enters the real number 7.3, this number and its integer value, `intval(7.3)`, are not equal.

Now, suppose that variable `$x` contains a value entered by the user. The code fragment, without data validation, would look like this.

```
if ($x == intval($x)) {
    echo $x, " is integer";
}
else {
    echo $x, " is real";
}
```

And now it's time to handle data validation. All you must do is combine the previous PHP script with the following one.

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));

if (is_numeric($x) == true) {
    //Here goes the previous code fragment
}
else {
    echo "Invalid number";
}
?>
```

Thus, the final script looks like this.

file_23.1-3.php

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));

if (is_numeric($x) == true) {
    if ($x == intval($x)) {
        echo $x, " is integer";
    }
}
```

```
}  
else {  
    echo $x, " is real";  
}  
}  
else {  
    echo "Invalid Number";  
}  
?>
```

Exercise 23.1-4 Converting Gallons to Liters, and Vice Versa

Write a PHP script that displays the following menu:

1. Convert gallons to liters
2. Convert liters to gallons

The script prompts the user to enter a choice (1 or 2) and a quantity. Then, it calculates and displays the required value. It is given that

$$1 \text{ gallon} = 3.785 \text{ liters}$$

Solution

The PHP script is shown here.

file_23.1-4.php

```
<?php  
define("COEFFICIENT", 3.785);  
  
echo "1: Gallons to liters\n";  
echo "2: Liters to gallons\n";  
echo "Enter choice: ";  
$choice = trim(fgets(STDIN));  
  
echo "Enter quantity: ";  
$quantity = trim(fgets(STDIN));  
  
if ($choice == 1) {  
    $result = $quantity * COEFFICIENT;  
    echo $quantity, " gallons equal to ", $result, " liters";  
}  
else {  
    $result = $quantity / COEFFICIENT;  
    echo $quantity, " liters equal to ", $result, " gallons";  
}
```

```
}  
?>
```

Exercise 23.1-5 *Converting Gallons to Liters, and Vice Versa (with Data Validation)*

Rewrite the PHP script of the previous exercise to validate the data input. Individual error messages should be displayed when the user enters a choice other than 1 or 2, or a non-numeric gas quantity.

Solution

Let's use the "from inner to outer" method. The inner code fragment has already been discussed in the previous exercise, whereas the outer PHP script to validate data input is shown here.

```
<?php  
define("COEFFICIENT", 3.785);  
  
echo "1: Gallons to liters\n";  
echo "2: Liters to gallons\n";  
echo "Enter choice: ";  
$choice = trim(fgets(STDIN));  
  
echo "Enter quantity: ";  
$quantity = trim(fgets(STDIN));  
  
if ($choice < 1 || $choice > 2) {  
    echo "Wrong choice!";  
}  
elseif (is_numeric($quantity) != true) {  
    echo "Invalid quantity!";  
}  
else {  
    //Here goes the code that converts  
    //gallons to liters and vice versa  
}  
?>
```

Combining this script with the one from the previous exercise, the final PHP script becomes

file_23.1-5.php

```
<?php  
define("COEFFICIENT", 3.785);
```



```

echo "1: Gallons to liters\n";
echo "2: Liters to gallons\n";
echo "Enter choice: ";
$choice = trim(fgets(STDIN));

echo "Enter quantity: ";
$quantity = trim(fgets(STDIN));

if ($choice < 1 || $choice > 2) {
    echo "Wrong choice!";
}
elseif (is_numeric($quantity) != true) {
    echo "Invalid quantity!";
}
else {
    if ($choice == 1) {
        $result = $quantity * COEFFICIENT;
        echo $quantity, " gallons equal to ", $result, " liters";
    }
    else {
        $result = $quantity / COEFFICIENT;
        echo $quantity, " liters equal to ", $result, " gallons";
    }
}
?>

```

From previous exercise

Exercise 23.1-6 *Where is the Tollkeeper?*

In a toll gate, there is an automatic system that recognizes whether the passing vehicle is a motorcycle, a car, or a truck. Write a PHP script that lets the user enter the type of the vehicle (M for motorcycle, C for car, and T for truck) and then displays the corresponding amount of money the driver must pay according to the following table.

Vehicle Type	Amount to Pay
Motorcycle	\$1
Car	\$2
Track	\$4

If the user enters a character other than M, C, or T, a corresponding error message should be displayed.

(Some more practice with data validation!)

Solution

The solution to this problem is quite simple. The only catch is that the user may enter the uppercase letters M, C, or T, or even the lowercase letters m, c, or t. The script needs to accept both. To handle this, you can convert the user's input to uppercase using the `strtoupper()` function. Then you need to test only for the M, C, or T characters in uppercase.

The PHP script is shown here.

file_23.1-6.php

```
<?php
    $v = trim(fgets(STDIN));

    $v = strtoupper($v);           //This function converts anything
                                   //to uppercase

    if ($v == "M") {               //You need to test only for capital M
        echo "You need to pay $1";
    }
    elseif ($v == "C") {           //You need to test only for capital C
        echo "You need to pay $2";
    }
    elseif ($v == "T") {           //You need to test only for capital T
        echo "You need to pay $4";
    }
    else {
        echo "Invalid vehicle";
    }
?>
```

Exercise 23.1-7 *The Most Scientific Calculator Ever!*

*Write a PHP script that prompts the user to enter a number, the type of operation (+, -, *, /), and a second number. The script should execute the required operation and display the result.*

Solution

The only thing that you need to take care of in this exercise is the possibility that the user could enter zero for the divisor (the second number), and as you know from mathematics, division by zero is undefined.

The following PHP script uses the case decision structure to check the type of operation.

file_23.1-7.php

```
<?php
echo "Enter 1st number: ";
$a = trim(fgets(STDIN));
echo "Enter type of operation: ";
$op = trim(fgets(STDIN));
echo "Enter 2nd number: ";
$b = trim(fgets(STDIN));

switch ($op) {
    case "+":
        echo $a + $b;
        break;
    case "-":
        echo $a - $b;
        break;
    case "*":
        echo $a * $b;
        break;
    case "/":
        if ($b == 0) {
            echo "Error: Division by zero";
        }
        else {
            echo $a / $b;
        }
        break;
}

?>
```

23.2 Decision Control Structures in Solving Mathematical Problems

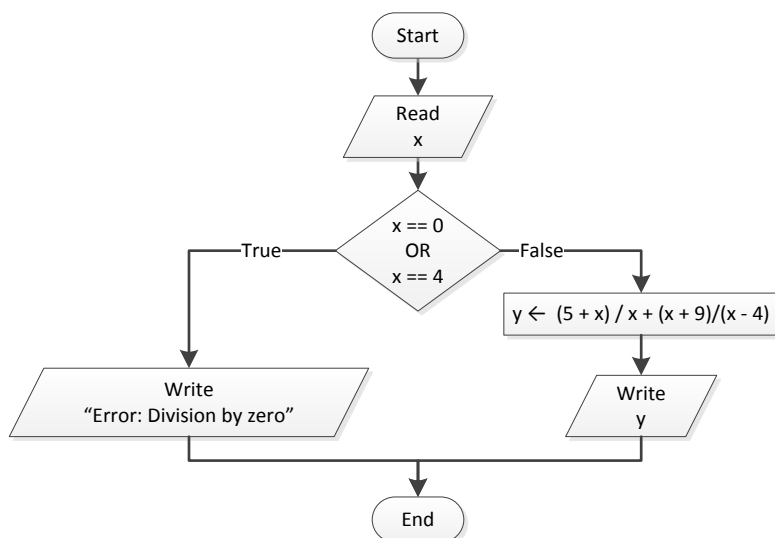
Exercise 23.2-1 Finding the Value of y

Design a flowchart and write the corresponding PHP script that finds and displays the value of y (if possible) in the following formula.

$$Y = \frac{5 + x}{x} + \frac{x + 9}{x - 4}$$

Solution

In this exercise the user should not be allowed to enter values 0 or 4 because they make the denominator equal to zero. Therefore, the script needs to take these restrictions into consideration. The flowchart is shown here.



and the PHP script is shown here.

file_23.2-1.php

```

<?php
    $x = trim(fgets(STDIN));

    if ($x == 0 || $x == 4) {
        echo "Error: Division by zero!";
    }
    else {
        $y = (5 + $x) / $x + ($x + 9) / ($x - 4);
        echo $y;
    }
  
```

?>

Exercise 23.2-2 Finding the Values of y

Design a flowchart and write the corresponding PHP script that finds and displays the values of y (if possible) in the following formula.

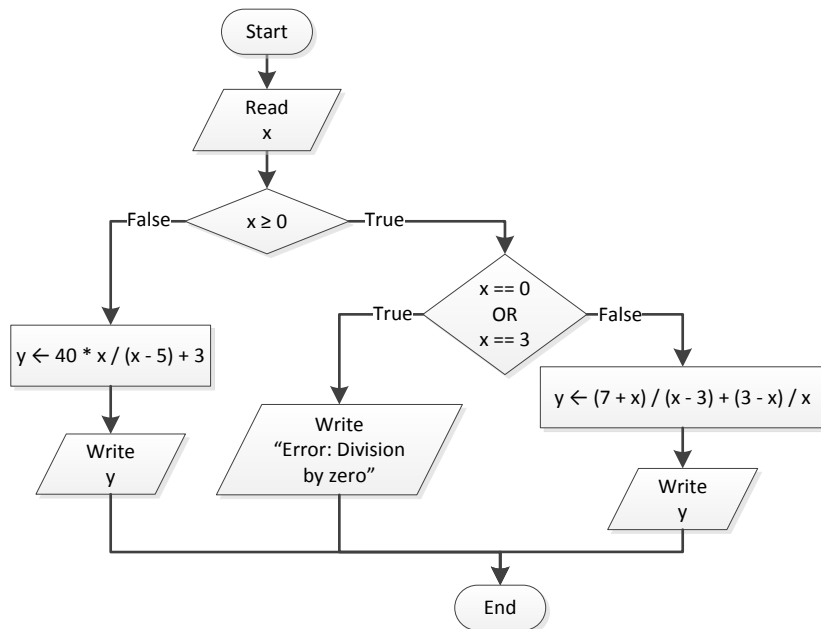
$$y = \begin{cases} \frac{7+x}{x-3} + \frac{3-x}{x}, & x \geq 0 \\ \frac{40x}{x-5} + 3, & x < 0 \end{cases}$$

Solution

The formula has two different results.

1. When x is greater than or equal to zero, the value of y in $\frac{7+x}{x-3} + \frac{3x}{x}$ can be found following the method shown in the previous exercise.
2. However, for an x less than zero, a small detail can save you some lines of PHP code. If you look carefully, you can see that there are no restrictions on the fraction $\frac{40x}{x-5}$ because x can never be +5.

The flowchart is shown here.



The PHP script is shown here.

file_23.2-2.php

```
<?php
    $x = trim(fgets(STDIN));

    if ($x >= 0 ) {
        if ($x == 0 || $x == 3) {
            echo "Error: Division by zero!";
        }
        else {
            $y = (7 + $x) / ($x - 3) + (3 - $x) / $x;
            echo $y;
        }
    }
    else {
        $y = 40 * $x / ($x - 5) + 3;
        echo $y;
    }
?>
```

Exercise 23.2-3 *Validating Data Input and Finding the Values of y*

Rewrite the PHP script of the previous exercise to validate the data input. An error message should be displayed when the user enters non-numeric characters.

(Again, you will get more practice with data validation!)

Solution

All you need to do is combine the previous exercise with the following PHP script.

```
<?php
    $x = trim(fgets(STDIN));

    if (is_numeric($x) == true) {
        //Here goes the code of the previous exercise
    }
    else {
        echo "Invalid number";
    }
?>
```

Combining this with the script of the previous exercise, the final script becomes

file_23.2-3.php

<?php

`$x = trim(fgets(STDIN));``if (is_numeric($x) == true) {``if ($x >= 0) {``if ($x == 0 || $x == 3) {``echo "Error: Division by zero!";``}``else {``$y = (7 + $x)/($x - 3) + (3 - $x)/$x;``echo $y;``}``}``else {``$y = 40*$x/($x-5)+3;``echo $y;``}``}``else {``echo "Invalid number";``}``?>`

From previous exercise

Exercise 23.2-4 Solving the Linear Equation $ax + b = 0$

Design a flowchart and write the corresponding PHP script that finds and displays the root of the linear equation

$$ax + b = 0$$

Solution

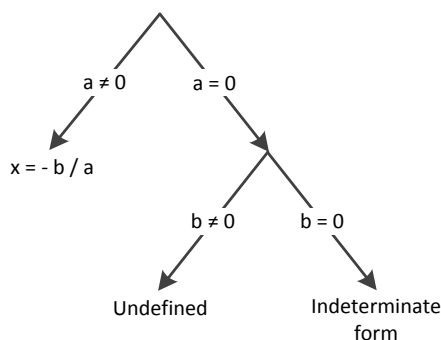
The equation $ax + b = 0$, when solved for x , becomes $x = -b / a$. The user must enter values for coefficients a and b , and the program should find the value for x .

Yet, you need to realize that, depending on user's entered data, three possible situations can arise:

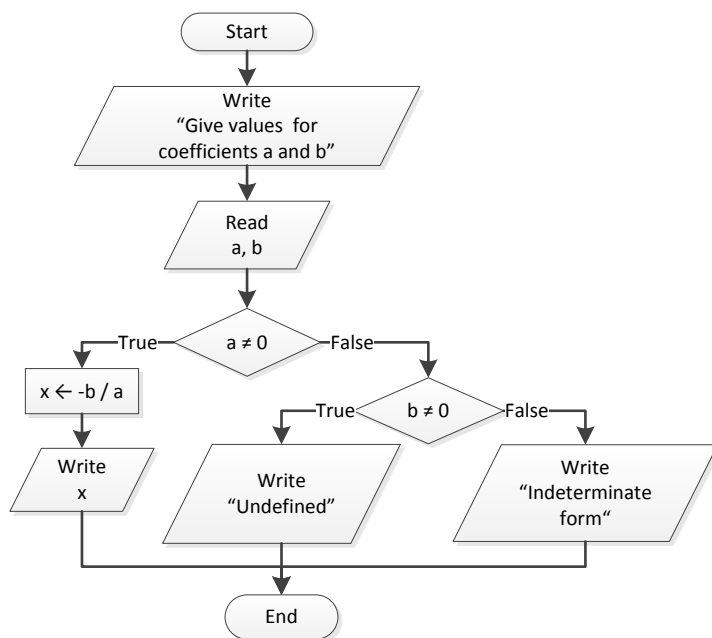
- i. The user might enter the value 0 for coefficient a and a non-zero value for coefficient b . The division by zero, as you already know from mathematics, is undefined.
- ii. The user might enter the value 0 for coefficient a and the value 0 for coefficient b . The result has no defined value, and it is called indeterminate form.

iii. The user might enter any other pair of values.

Therefore, these three situations result in three paths, respectively.



The next flowchart is designed with the use of a multiple-alternative decision structure.



The PHP script is shown here.

file_23.2-4.php

```
<?php
```

```

echo "Enter a value for coefficient a: ";
$a = trim(fgets(STDIN));
echo "Enter a value for coefficient b: ";
$b = trim(fgets(STDIN));
  
```



```

if ($a != 0) {
    $x = -$b / $a;
    echo $x;
}
elseif ($b != 0) {
    echo "Undefined";
}
else {
    echo "Indeterminate form";
}
?>

```

Exercise 23.2-5 Solving the Quadratic Equation $ax^2 + bx + c = 0$

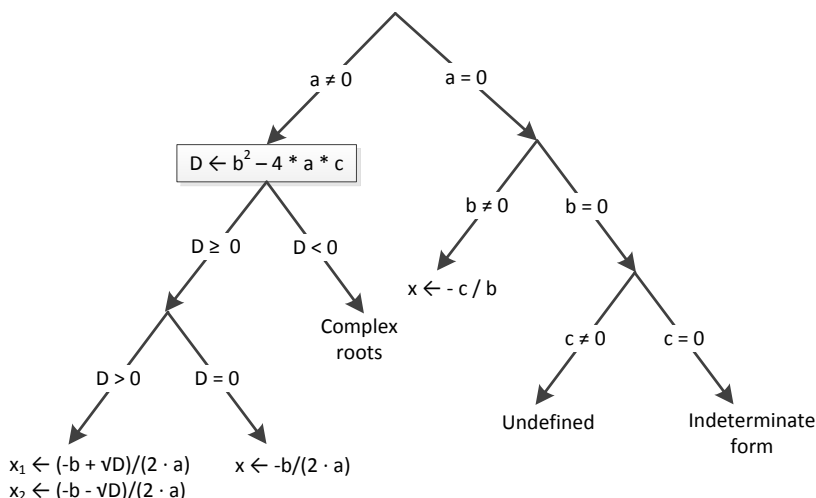
Design a flowchart and write the corresponding PHP script that finds and displays the roots of the quadratic equation

$$ax^2 + bx + c = 0$$

Solution

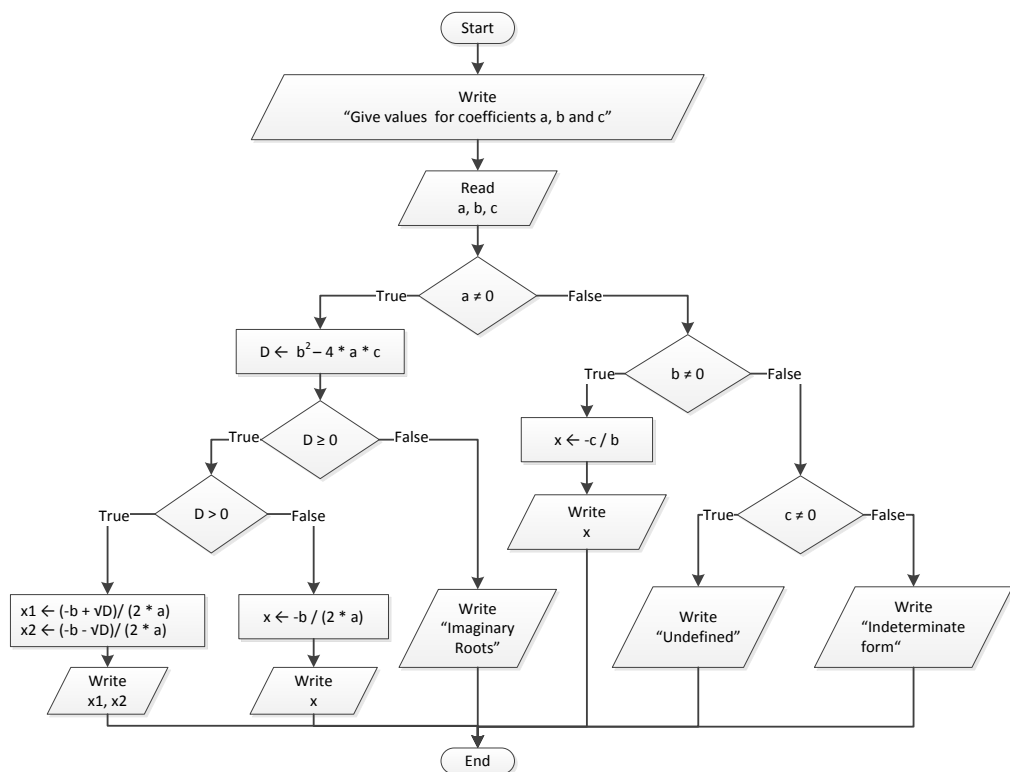
This problem can be divided into two individual subproblems depending on the value of coefficient a . If coefficient a is equal to zero, the equation becomes a linear equation, $bx + c = 0$, for which the solution was given in the previous exercise. If coefficient a is not equal to zero, the roots of the equation can be found using the discriminant D . Please note that this solution finds no complex roots when $D < 0$; this is beyond the scope of this book.

All necessary paths are as follows.



Notice: The path on the right ($a = 0$) is the solution to the linear equation $bx + c = 0$, which was shown in the previous exercise.

Using this diagram you can design the following flowchart.



The PHP script is shown here.

file_23.2-5.php

```
<?php
```

```
echo "Enter values for coefficients a, b and c: ";
```

```
$a = trim(fgets(STDIN));
```

```
$b = trim(fgets(STDIN));
```

```
$c = trim(fgets(STDIN));
```

```
if ($a != 0) {
```

```
    $D = $b * $b - 4 * $a * $c;
```

```
    if ($D >= 0) {
```

```
        if ($D > 0) {
```

```
            $x1 = (-$b + sqrt($D)) / (2 * $a);
```

```
    $x2 = (-$b - sqrt($D)) / (2 * $a);
    echo "Roots: " , $x1 , ", " , $x2;
}
else {
    $x = -$b / (2 * $a);
    echo "One double root: " , $x;
}
}
else {
    echo "Complex Roots";
}
}
else {
    if ($b != 0) {
        $x = -$c / $b;
        echo "Root: " , $x;
    }
    elseif ($c != 0) {
        echo "Undefined";
    }
    else {
        echo "Indeterminate form";
    }
}
}
?>
```

23.3 Finding Minimum and Maximum Values with Decision Control Structures

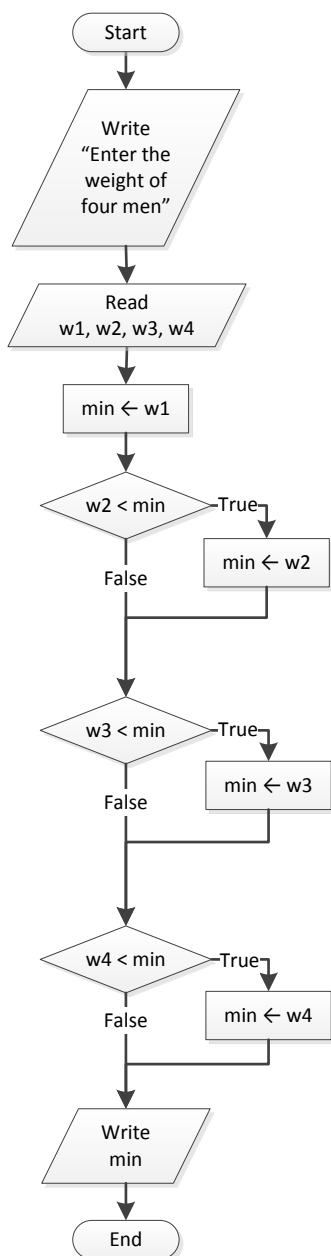
Suppose there are some men and you want to find the lightest one. Let's say that each one of them comes by and tells you his weight. What you must do is, memorize the weight of the first person that has come by and for each new person, you have to compare his weight with the one that you keep memorized. If he is heavier, you ignore his weight. However, if he is lighter, you need to forget the previous weight and memorize the new one. The same procedure continues until all the people have come by.

Let's ask four men to come by at a random order. Assume that their weights, in order of appearance, are 165, 170, 160, and 180 pounds.

Procedure	Value of Variable <code>min</code> in Your Mind!
The first person comes by. He weighs 165 pounds. Keep his weight in your mind (imagine a variable in your mind named <code>min</code> .)	<code>min = 165</code>
The second person comes by. He weighs 170 pounds. He does not weigh less than the weight you are keeping in variable <code>min</code> , so you must ignore his weight. Variable <code>min</code> in your mind still contains the value 165.	<code>min = 165</code>
The third person comes by. He weighs 160 pounds, which is less than the weight you are keeping in variable <code>min</code> , so you must forget the previous value and keep the value 160 in variable <code>min</code> .	<code>min = 160</code>
The fourth person comes by. He weighs 180 pounds. He does not weigh less than the weight you are keeping in variable <code>min</code> , so you must ignore his weight. Variable <code>min</code> still contains the value 160.	<code>min = 160</code>

When the procedure finishes, the variable `min` in your mind contains the weight of the lightest man!

Following are the flowchart and the corresponding PHP script that prompts the user to enter the weight of four people and then finds and displays the lightest weight.



file_23.3.php

```
<?php
```

```
    echo "Enter the weight of four men:";
```

```
    $w1 = trim(fgets(STDIN));
```

```
    $w2 = trim(fgets(STDIN));
```

```
    $w3 = trim(fgets(STDIN));
```

```
    $w4 = trim(fgets(STDIN));
```

```
    //memorize the weight of the first person
```

```
    $min = $w1;
```

```
    //if second one is lighter, forget
```

```
    //everything and memorize this weight
```

```
    if ($w2 < $min) {
```

```
        $min = $w2;
```

```
    }
```

```
    //if third one is lighter, forget
```

```
    //everything and memorize this weight
```

```
    if ($w3 < $min) {
```

```
        $min = $w3;
```

```
    }
```

```
    //if fourth one is lighter, forget
```

```
    //everything and memorize this weight
```

```
    if ($w4 < $min) {
```

```
        $min = $w4;
```

```
    }
```

```
    echo $min;
```

```
?>
```

Notice: You can find the maximum instead of the minimum value by simply replacing the "less than" with a "greater than" operator in all Boolean expressions.

Exercise 23.3-1 *Finding the Name of the Heaviest Person*

Write a PHP script that prompts the user to enter the weights and the names of three people and then displays the name and the weight of the heaviest person.

Solution

In this exercise, along with the maximum weight, you need to store in another variable the name of the person who actually has that weight. The PHP script is shown here.

file_23.3-1.php

```
<?php
echo "Enter the weight of the first person: ";
$w1 = trim(fgets(STDIN));

echo "Enter the name of the first person: ";
$n1 = trim(fgets(STDIN));

echo "Enter the weight of the second person: ";
$w2 = trim(fgets(STDIN));

echo "Enter the name of the second person: ";
$n2 = trim(fgets(STDIN));

echo "Enter the weight of the third person: ";
$w3 = trim(fgets(STDIN));

echo "Enter the name of the third person: ";
$n3 = trim(fgets(STDIN));

$max = $w1;
$m_name = $n1;      //This variable holds the name of the heaviest person

if ($w2 > $max) {
    $max = $w2;
    $m_name = $n2;   //Someone else is heavier. Keep his or her name.
}

if ($w3 > $max) {
    $max = $w3;
    $m_name = $n3;   //Someone else is heavier. Keep his or her name.
```

```
}

echo "The heaviest person is ", $m_name;
echo "\nHis or her weight is ", $max;

?>
```

Notice: In case the two heaviest people happen to have the same weight, the name of the first one in order is found and displayed.

23.4 Exercises with Series of Consecutive Ranges of Values

As you have already seen, in many problems the value of a variable or the result of an expression can define which statement or block of statements should be executed. In the exercises that follow, you will learn how to test if a value or the result of an expression belongs within a specific range of values (from a series of consecutive ranges of values).

Suppose that you want to display a message indicating the types of clothes a woman might wear at different temperatures.

Outdoor Temperature (in degrees Fahrenheit)	Types of Clothes a Woman Might Wear
Temperature < 45	Sweater, coat, jeans, shirt, shoes
45 ≤ Temperature < 65	Sweater, jeans, jacket, shoes
65 ≤ Temperature < 75	Capris, shorts, t-shirt, tank top, flip flops, athletic shoes
75 ≤ Temperature	Shorts, t-shirt, tank top, skort, skirt, flip flops

At first glance you might be tempted to use single-alternative decision structures. It is not wrong actually but if you take a closer look, it becomes clear that each condition is interdependent, which means that when one of these evaluates to `true`, none of the others should be evaluated. You need to select just one alternative from a set of possibilities.

There are actually two decision control structures that can be used for this purpose, and these are the multiple-alternative decision structure and nested decision control structures. However, the multiple-alternative decision structure is the best choice. It is more convenient and increases readability.

Exercise 23.4-1 Calculating the Discount

Write a PHP script that calculates the discount that customers receive based on the dollar amount of their order. If the total amount ordered is less than \$30, no discount is given. If the total amount is equal to or greater than \$30 and less than \$70, a discount of 5% is given. If the total amount is equal to or greater than \$70 and less than \$150, a discount of 10% is given. If the total amount is \$150 or more, the customer receives a discount of 20%.

Solution

The following table summarizes the various discounts that are offered.

Range	Discount
$\text{amount} < \$30$	0%
$\$30 \leq \text{amount} < \$70$	5%
$\$70 \leq \text{amount} < \$150$	10%
$\$150 \leq \text{amount}$	20%

The PHP script is shown here.

file_23.4-1a.php

```
<?php
echo "Enter total amount: ";
$amount = trim(fgets(STDIN));

if ($amount < 30) {
    $discount = 0;
}
elseif ($amount >= 30 && $amount < 70) {
    $discount = 5;
}
elseif ($amount >= 70 && $amount < 150) {
    $discount = 10;
}
elseif ($amount >= 150) {
    $discount = 20;
}

$payment = $amount - $amount * $discount / 100;

echo "You got a discount of ", $discount, "%\n";
echo "You must pay $", $payment;

?>
```

A closer examination, however, reveals that the Boolean expressions written in bold are not actually required. For example, when the first Boolean expression evaluates to false, the flow of execution continues to evaluate the second Boolean expression, in which, variable `$amount` is definitely greater than or equal to 30. Therefore, the Boolean expression `$amount >= 30`, when evaluated, is certainly true and thus can be omitted. The same logic applies to all cases; you can omit all Boolean expressions

written in bold. The final PHP script is shown here, with all unnecessary evaluations removed.

file_23.4-1b.php

```
<?php
echo "Enter total amount: ";
$amount = trim(fgets(STDIN));

if ($amount < 30) {
    $discount = 0;
}
elseif ($amount < 70) {
    $discount = 5;
}
elseif ($amount < 150) {
    $discount = 10;
}
else {
    $discount = 20;
}

$payment = $amount - $amount * $discount / 100;

echo "You got a discount of ", $discount, "%\n";
echo "You must pay $", $payment;
?>
```

Exercise 23.4-2 Validating Data Input and Calculating the Discount

Rewrite the PHP script of the previous exercise to validate the data input. Individual error messages should be displayed when the user enters any non-numeric or negative values.

Solution

Let's try once again the “from inner to outer” method. The inner code fragment has already been discussed in the previous exercise, whereas the outer PHP script that validates data input is shown here.

```
<?php
echo "Enter total amount: ";
$amount = trim(fgets(STDIN));

if (is_numeric($amount) != true) {
```

```
    echo "Entered value contains non-numeric characters";
}
elseif ($amount < 0) {
    echo "Entered value is negative";
}
else {
    //Here goes the code that calculates
    //and displays the discount offered
}
?>
```

Combining this with the script of the previous exercise, the final script becomes

file_23.4-2.php

```
<?php
echo "Enter total amount: ";
$amount = trim(fgets(STDIN));

if (is_numeric($amount) != true) {
    echo "Entered value contains non-numeric characters";
}
elseif ($amount < 0) {
    echo "Entered value is negative";
}
else {
    if ($amount < 30) {
        $discount = 0;
    }
    elseif ($amount < 70) {
        $discount = 5;
    }
    elseif ($amount < 150) {
        $discount = 10;
    }
    else {
        $discount = 20;
    }

    $payment = $amount - $amount * $discount / 100;

    echo "You got a discount of ", $discount, "%\n";
}
```

From previous exercise



```
| echo "You must pay $", $payment;
|
| }
|
?>
```

Exercise 23.4-3 *Sending a Parcel*

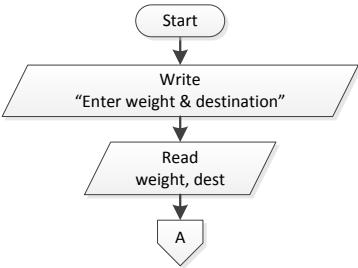
In a post office, the shipping cost for sending a medium parcel depends on its weight and whether its destination is inside or outside the country. Shipping costs are calculated according to the following table.

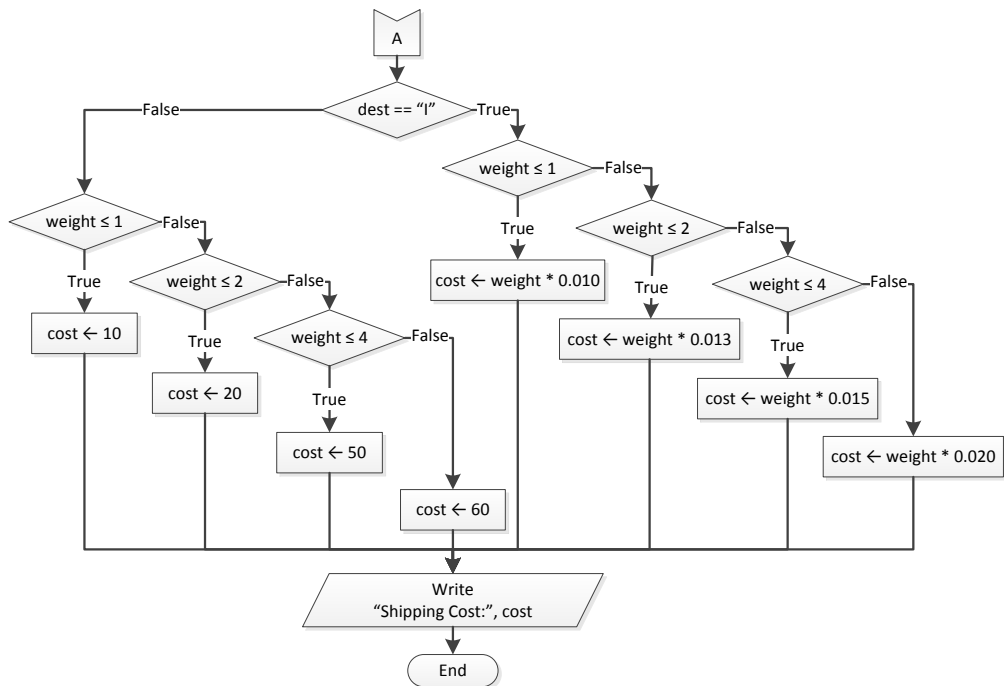
Parcel's Weight (in lb)	Destination Inside the Country (in USD per lb)	Destination Outside the Country (in USD)
weight < 1	\$0.010	\$10
1 ≤ weight < 2	\$0.013	\$20
2 ≤ weight < 4	\$0.015	\$50
4 ≤ weight	\$0.020	\$60

Design a flowchart and write the corresponding PHP script that prompts the user to enter the weight of a parcel and its destination (I: inside the country, O: outside the country) and then calculates and displays the shipping cost.

Solution

Here is the flowchart.





The PHP script is shown here.

file_23.4-3.php

```
<?php
```

```
echo "Enter weight and destination: ";
```

```
$weight = trim(fgets(STDIN));
```

```
$dest = trim(fgets(STDIN));
```

```
if (strtoupper($dest) == "I") {
```

```
    if ($weight <= 1) {
```

```
        $cost = $weight * 0.010;
```

```
    }
```

```
    elseif ($weight <= 2) {
```

```
        $cost = $weight * 0.013;
```

```
    }
```

```
    elseif ($weight <= 4) {
```

```
        $cost = $weight * 0.015;
```

```
    }
```

```
    else {
```

```
        $cost = $weight * 0.020;
```

```
    }
```

```
}
```

```

else {
    if ($weight <= 1) {
        $cost = 10;
    }
    elseif ($weight <= 2) {
        $cost = 20;
    }
    elseif ($weight <= 4) {
        $cost = 50;
    }
    else {
        $cost = 60;
    }
}

echo "Shipping cost: ", $cost;

?>

```

Notice: A user may enter the letter I (for destination) in lowercase or uppercase. The function `strtoupper()` ensures that the script executes properly for both cases.

Exercise 23.4-4 Finding the Values of y

Design a flowchart and write the corresponding PHP script that finds and displays the values of y (if possible) in the following formula

$$y = \left\{ \begin{array}{ll} \frac{x}{x-3} + \frac{8+x}{x+1}, & -5 < x \leq 0 \\ \frac{40x}{x-8}, & 0 < x \leq 6 \\ \frac{3x}{x-9}, & 6 < x \leq 20 \\ |x|, & \text{for all other values of } x \end{array} \right\}$$

Solution

In this exercise, there are two restrictions on the fractions:

1. In fraction $\frac{8+x}{x+1}$, the value of x cannot be -1 .
2. In fraction $\frac{3x}{x-9}$, the value of x cannot be $+9$.

For all other fractions, it's impossible for the denominators to be set to zero because of the range in which x belongs.

The PHP script is shown here.

file_23.4-4a.php

```
<?php
```

```
echo "Enter a value for x: ";
```

```
$x = trim(fgets(STDIN));
```

```
if ($x > -5 && $x <= 0) {
```

```
    if ($x != -1) {
```

```
        $y = $x / ($x - 3) + (8 + $x) / ($x + 1);
```

```
        echo $y;
```

```
    }
```

```
    else {
```

```
        echo "Invalid value";
```

```
    }
```

```
}
```

```
elseif ($x > 0 && $x <= 6) {
```

```
    $y = 40 * $x / ($x - 8);
```

```
    echo $y;
```

```
}
```

```
elseif ($x > 6 && $x <= 20) {
```

```
    if ($x != 9) {
```

```
        $y = 3 * $x / ($x - 9);
```

```
        echo $y;
```

```
    }
```

```
    else {
```

```
        echo "Invalid value";
```

```
    }
```

```
}
```

```
else {
```

```
    $y = abs($x);
```

```
    echo $y;
```

```
}
```

```
?>
```

$-5 < x \leq 0$

$0 < x \leq 6$

$6 < x \leq 20$

All other values of x

If you are wondering whether you can remove all `echo $y` statements and write them as a single statement at the end of the script, the answer is “no.” Since there is even one path that does not include the `echo $y` statement, you must write the same statement, again and again, in every path required.

One thing you should learn in computer programming, however, is that you should never give up! By modifying your code a little, checking for invalid values in the

beginning, gives you the opportunity to remove the statement (`echo $y`) outside of all paths and move it at the end of them. The modified PHP script is shown here.

file_23.4-4b.php

```
<?php
    echo "Enter a value for x: ";
    $x = trim(fgets(STDIN));

    if ($x == -1 || $x == 9) {
        echo "Invalid value";
    }
    else {
        if ($x > -5 && $x <= 0) {
            $y = $x / ($x - 3) + (8 + $x) / ($x + 1);
        }
        elseif ($x > 0 && $x <= 6) {
            $y = 40 * $x / ($x - 8);
        }
        elseif ($x > 6 && $x <= 20) {
            $y = 3 * $x / ($x - 9);
        }
        else {
            $y = abs($x);
        }
        echo $y;
    }
?>
```

Now, if you are speculating about whether the Boolean expressions in bold are redundant, the answer is “no” again. Suppose you do remove them and the user enters a value of -20 for x . The flow of execution then reaches the Boolean expression $\$x \leq 0$, which evaluates to true. Then the fraction $\frac{x}{x-3} + \frac{8+x}{x+1}$, instead of the absolute value of x , is calculated.

Yet still there is hope! Next, you can find a proposed solution in which unnecessary Boolean expressions are actually removed. The trick here is that the case of the absolute value of x is examined first.

file_23.4-4c.php

```
<?php
    echo "Enter a value for x: ";
    $x = trim(fgets(STDIN));

    if ($x == -1 || $x == 9) {
```

```

    echo "Invalid value";
}
else {
    if ($x <= -5 || $x > 20) {
        $y = abs($x);
    }
    elseif ($x <= 0) {
        $y = $x / ($x - 3) + (8 + $x) / ($x + 1);
    }
    elseif ($x <= 6) {
        $y = 40 * $x / ($x - 8);
    }
    else {
        $y = 3 * $x / ($x - 9);
    }
    echo $y;
}
?>

```

Notice: It is obvious that one problem can have many solutions. It is up to you to find the optimal one!

Exercise 23.4-5 *Progressive Rates and Electricity Consumption*

The LAV Electricity Company charges subscribers for their electricity consumption according to the following table (monthly rates for domestic accounts). Assume that all extra charges such as transmission service charges and distribution charges are all included. Please note that the rates are progressive.

Kilowatt-hours (kWh)	USD per kWh
kWh $\leq$ 500	\$0.10
501 $\leq$ kWh $\leq$ 2000	\$0.25
2001 $\leq$ kWh $\leq$ 4000	\$0.40
4001 < kWh	\$0.60

Write a PHP script that prompts the user to enter the total number of kWh consumed and then calculates and displays the total amount to pay.

Solution

The term “progressive rates” means that when a customer consumes, for example, 2200 kWh, not all of the kilowatt-hours are charged at \$0.40. The first 500 kWh are

charged at \$0.10, the next 1500 kWh are charged at \$0.25 and only the last 200 kWh are charged at \$0.40. Thus, the customer must pay

$$500 \times 0.10 + 1500 \times 0.25 + 200 \times 0.40 = \$505$$

The same logic can be used to calculate the total amount to pay when the customer consumes, for example, 4300 kWh. The first 500 kWh are charged at \$0.10, the next 1500 kWh are charged at \$0.25, the next 2000 kWh are charged at 0.40, and only the last 300 kWh are charged at \$0.60. Thus, the customer must pay

$$500 \times 0.10 + 1500 \times 0.25 + 2000 \times 0.40 + 300 \times 0.60 = \$1405$$

The PHP script is shown here.

file_23.4-5.php

```
<?php
echo "Enter number of Kilowatt-hours consumed: ";
$kwh = trim(fgets(STDIN));

if ($kwh <= 500) {
    $t = $kwh * 0.10;
}
elseif ($kwh <= 2000) {
    $t = 500 * 0.10 + ($kwh - 500) * 0.25;
}
elseif ($kwh <= 4000) {
    $t = 500 * 0.10 + 1500 * 0.25 + ($kwh - 2000) * 0.40;
}
else {
    $t = 500 * 0.10 + 1500 * 0.25 + 2000 * 0.4 + ($kwh - 4000) * 0.60;
}

echo "Total amount to pay: ", $t;

?>
```

Exercise 23.4-6 *Progressive Rates, Electricity Consumption, Taxes, Data Validation and Code Optimization, All in One!*

Rewrite the PHP script of the previous exercise, to validate the data input. Individual error messages should be displayed when the user enters any non-numeric or negative values. Moreover, you need to optimize the script to calculate the results as quickly as possible.

Federal, state, and local taxes add a total of 10% to each bill.

Solution

To optimize the code, you need to decrease the total number of multiplication, addition, and subtraction operations that the previous script had. For example, the statement

```
$t = 500 * 0.10 + ($kwh - 500) * 0.25;
```

can also be written as

```
$t = 50 + ($kwh - 500) * 0.25;
```

or even better as

```
$t = 0.25 * $kwh - 75;
```

Similarly, the statement

```
$t = 500 * 0.10 + 1500 * 0.25 + ($kwh - 2000) * 0.40;
```

can also be written as

```
$t = 50 + 375 + ($kwh - 2000) * 0.40;
```

or even better as

```
$t = 425 + ($kwh - 2000) * 0.40;
```

and finally as

```
$t = 0.4 * $kwh - 375;
```

Now that everything about the inner code fragment has been clarified, data validation can be achieved by following the “from inner to outer” method. The outer PHP script given in general form that validates data input is

```
<?php
echo "Enter number of Kilowatt-hours consumed: ";
$kwh = trim(fgets(STDIN));

if (is_numeric($kwh) != true) {
    echo "Entered value contains non-numeric characters";
}
elseif ($kwh < 0) {
    echo "Entered value is negative";
}
else {
    //Here goes the code that calculates
    //and displays the total amount to pay
}
?>
```

Combining this script with the last script from the previous exercise and applying code optimization, the PHP script becomes

file_23.4-6.php

```
<?php
```

```

echo "Enter number of Kilowatt-hours consumed: ";
$kwH = trim(fgets(STDIN));

if (is_numeric($kwH) != true) {
    echo "Entered value contains non-numeric characters";
}
elseif ($kwH < 0) {
    echo "Entered value is negative";
}
else {
    if ($kwH <= 500) {
        $t = $kwH * 0.10;
    }
    elseif ($kwH <= 2000) {
        $t = 0.25 * $kwH - 75;
    }
    elseif ($kwH <= 4000) {
        $t = 0.4 * $kwH - 375;
    }
    else {
        $t = 0.6 * $kwH - 1175;
    }

    $t = 1.10 * $t;           //This is equivalent to
                             // $t = $t + $t * 10 / 100
                             //but more efficient

    echo "Total amount to pay (taxes included): ", $t;
}
?>

```

From previous exercise after
applying code optimization



Notice: It becomes clear now that, even though this script is more efficient than the one of the previous exercise, it is very difficult to read. No one who reads this script can understand that $t = 0.6 * kwH - 1175$ is actually $t = 500 * 0.10 + 1500 * 0.25 + 2000 * 0.4 + (kwH - 4000) * 0.60$. Many times, when efficiency is not important, programmers do sacrifice efficiency for readability. This script is shown to demonstrate how a script can be made more efficient. But you don't have to follow this kind of writing, at least not for now anyway. After all, today's computers are so fast that you don't have to worry about efficiency. Until the time at which you become the next Mark Zuckerberg of Facebook or the next Bill Gates of Microsoft, please write readable scripts! This book as well, since it addresses novice programmers, mostly prefers readability to efficiency.

Exercise 23.4-7 *Progressive Rates and Text Messaging Services*

The LAV Cell Phone Company charges customers a basic rate of \$8 per month to send text messages. Additional rates are charged based on the total number of text messages sent, as shown in the following table.

Number of Text Messages Sent	Additional Rates (in USD per text message)
Up to 50	Free of charge
51 – 150	\$0.05
151 and above	\$0.10

Federal, state, and local taxes add a total of 10% to each bill.

Write a PHP script that prompts the user to enter the number of text messages sent and then calculates and displays the total amount to pay.

Please note that the rates are progressive.

Solution

The PHP script is presented here.

file_23.4-7.php

```
<?php
echo "Enter number of text messages sent: ";
$count = trim(fgets(STDIN));

if ($count <= 50) {
    $extra = 0;
}
elseif ($count <= 150) {
    $extra = ($count - 50) * 0.05;
}
else {
    $extra = 100 * 0.05 + ($count - 150) * 0.10;
}

$total_without_tax = 8 + $extra;
$tax = $total_without_tax * 10 / 100;
$total = $total_without_tax + $tax;
echo "Total amount to pay: ", $total;
?>
```

23.5 Exercises of a General Nature with Decision Control Structures

Exercise 23.5-1 Finding a Leap Year

Write a PHP script that prompts the user to enter a year and then displays a message indicating whether or not it is a leap year. Moreover, if the user enters non-numeric characters, an error message should be displayed.

(Note that this involves data validation!)

Solution

A year is a leap year when at least one of the following conditions is met:

1. **1st Condition:** The year is exactly divisible by 4, and not by 100.
2. **2nd Condition:** The year is exactly divisible by 400.

In the following table, some years are not leap years because none of the conditions evaluates to `true`.

Year	Conditions	Leap Year
1600	1 st Condition: False, because it is exactly divisible by 4, and also by 100	Yes
	2 nd Condition: True , because it is exactly divisible by 400	
1900	1 st Condition: False, because it is exactly divisible by 4, and also by 100	No
	2 nd Condition: False, because it is not exactly divisible by 400	
1918	1 st Condition: False, because it is not exactly divisible by 4	No
	2 nd Condition: False, because it is not exactly divisible by 400	
2000	1 st Condition: False, because it is exactly divisible by 4, and also by 100	Yes
	2 nd Condition: True , because it is exactly divisible by 400	
2002	1 st Condition: False, because it is not exactly divisible by 4	No
	2 nd Condition: False, because it is not exactly divisible by 400	
2004	1 st Condition: True , because it is exactly divisible by 4, and not by 100	Yes
	2 nd Condition: False, because it is not exactly divisible by 400	
2016	1 st Condition: True , because it is exactly divisible by 4, and not by 100	Yes
	2 nd Condition: False, because it is not exactly divisible by 400	

The PHP script is shown here.

file_23.5-1.php

```
<?php
echo "Enter a year: ";
$y = trim(fgets(STDIN));

if (is_numeric($y) == true) {
    if ($y % 4 == 0 && $y % 100 != 0 || $y % 400 == 0) {
        echo "Leap year!";
    }
    else {
        echo "Not a leap year";
    }
}
else {
    echo "Invalid Number";
}
?>
```

Remember! The AND (&&) operator has a higher precedence than the OR (||) operator.

Exercise 23.5-2 *Displaying the Days of the Month*

Write a PHP script that prompts the user to enter a month and a year and then displays how many days are in that month. The script needs to take into consideration the leap years. In case of a leap year, February has 29 instead of 28 days.

Solution

The PHP script is shown here.

file_23.5-2a.php

```
<?php
echo "Enter month 1 - 12: ";
$m = trim(fgets(STDIN));
echo "Enter year: ";
$y = trim(fgets(STDIN));

if ($m == 2) {
    if ($y % 4 == 0 && $y % 100 != 0 || $y % 400 == 0) {
        echo "This month has 29 days";
    }
}
```

```
    else {
        echo "This month has 28 days";
    }
}
elseif ($m == 4 || $m == 6 || $m == 9 || $m == 11) {
    echo "This month has 30 days";
}
else {
    echo "This month has 31 days";
}
?>
```

Below, the same problem is solved again, using, however, the case decision structure.

file_23.5-2b.php

```
<?php
echo "Enter month 1-12: ";
$m = trim(fgets(STDIN));
echo "Enter year: ";
$y = trim(fgets(STDIN));

switch ($m) {
    case 2:
        if ($y % 4 == 0 && $y % 100 != 0 || $y % 400 == 0) {
            echo "This month has 29 days";
        }
        else {
            echo "This month has 28 days";
        }
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        echo "This month has 30 days";
        break;
    default:
        echo "This month has 31 days";
}
?>
```

Notice: Please note the way cases 4, 6, and 9 are written. Since there isn't any break statement in any of those cases, they all reach case 11.

Exercise 23.5-3 Is the Number a Palindrome?

A *palindrome* is a number that remains the same after reversing its digits. For example, the number 13631 is a palindrome. Write a PHP script that lets the user enter a five-digit integer and tests whether or not this number is a palindrome. Moreover, individual error messages should be displayed when the user enters any non-numeric characters or any integer with either less than or more than five digits.

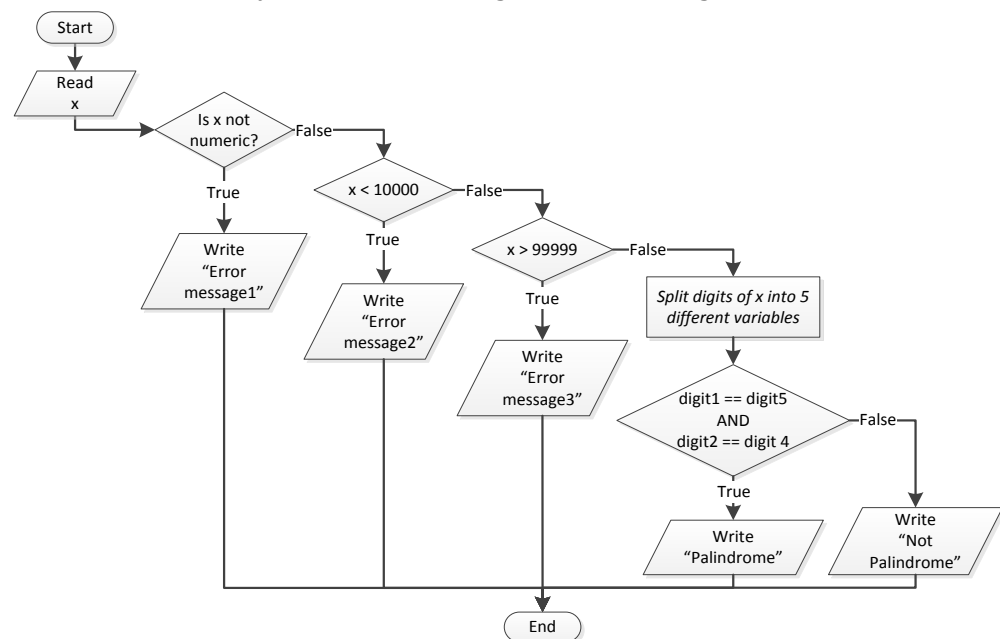
(Note that this involves data validation!)

Solution

To test if the user enters a palindrome number, you need to split its digits into five different variables as you learned in Chapter 13. Then, you can check whether the 1st digit is equal to the 5th digit and the 2nd digit is equal to the 4th digit. If this evaluates to true, the number is a palindrome.

To validate data input, you need to check whether the user has entered a five-digit number. Keep in mind that all five-digit numbers are in the range of 10000 to 99999. Therefore, you can just restrict the data input to within this range.

In order to display many different error messages, the best practice is to use a multiple-alternative decision structure which first checks data input validity for all cases, and then tries to solve the required problem. For example, if you need to check for various errors, you can do something like the following.



The final PHP script is shown here.

file_23.5-3.php

```
<?php
    $x = trim(fgets(STDIN));

    if (is_numeric($x) != true) {
        echo "You entered non-numeric characters";
    }
    elseif ($x < 10000) {
        echo "You entered less than five digits";
    }
    elseif ($x > 99999) {
        echo "You entered more than five digits";
    }
    else {
        $digit1 = intval($x / 10000);
        $r = $x % 10000;
        $digit2 = intval($r / 1000);
        $r = $r % 1000;
        $digit3 = intval($r / 100);
        $r = $r % 100;
        $digit4 = intval($r / 10);
        $digit5 = $r % 10;

        if ($digit1 == $digit5 && $digit2 == $digit4) {
            echo "Palindrome";
        }
        else {
            echo "Not Palindrome";
        }
    }
?>
```

Exercise 23.5-4 *Checking for Proper Capitalization and Punctuation*

Write a PHP script that prompts the user to enter a sentence and then checks it for proper capitalization and punctuation. The script should determine if the string begins with an uppercase letter and ends with a punctuation mark (check only for periods, question marks, and exclamation marks).

Solution

In this exercise you need to isolate the first and the last character of the string. As you already know, you can access any individual character of a string using substring notation. You can use index 0 to access the first character, index 1 to access the second character, and so on.

Thus, you can isolate the first character of a string using the following PHP statement.

```
$first_char = $sentence[0];
```

On the other hand, the index of the last character is 1 less than the length of the string. But, how long is that string?

You can find the length of any string using the `strlen()` function. The following code fragment isolates the last character of a string.

```
$length = strlen($sentence);  
$last_char = $sentence[$length - 1];
```

or you can even write it with one single statement.

```
$last_char = $sentence[strlen($sentence) - 1];
```

The PHP script is shown here.

file_23.5-4.php

```
<?php  
echo "Enter a sentence: ";  
$sentence = trim(fgets(STDIN));  
  
$first_char = $sentence[0];  
$last_char = $sentence[strlen($sentence) - 1];  
  
$sentence_is_okay = true;  
  
if ($first_char != strtoupper($first_char)) {  
    $sentence_is_okay = false;  
}  
elseif ($last_char != "." && $last_char != "?" && $last_char != "!") {  
    $sentence_is_okay = false;  
}  
  
if ($sentence_is_okay == true) {  
    echo "Sentence is okay!";  
}  
?>
```

In the beginning, the script assumes that the sentence is okay (`$sentence_is_okay = true`). Then, it checks for proper capitalization and proper

punctuation and if something goes wrong, it assigns the value `false` to the variable `$sentence_is_okay`.

23.6 Review Exercises

Complete the following exercises.

- Design a flowchart and write the corresponding PHP script that lets the user enter two integers and then displays a message indicating whether at least one integer is odd; otherwise, a message “Nothing special” is displayed. Moreover, if the user enters non-numeric values, an error message should be displayed.
- Design a flowchart and write the corresponding PHP script that lets the user enter two integers and then displays a message indicating whether or not they are both divisible exactly by 3 and by 4. For example, 12 is such a number. Moreover, individual error messages should be displayed when the user enters any non-numeric values or any floats.
- Write a PHP script that displays the following menu:
 - Convert Kelvin to Fahrenheit
 - Convert Fahrenheit to Kelvin
 - Convert Fahrenheit to Celsius
 - Convert Celsius to Fahrenheit

It then prompts the user to enter a choice (of 1 to 4) and a temperature value. It then calculates and displays the required value. Moreover, individual error messages should be displayed when the user enters a choice other than 1, 2, 3, or 4, or a non-numeric temperature value.

It is given that

$$1.8 \times \text{Kelvin} = \text{Fahrenheit} + 459.67$$

and

$$\frac{\text{Celsius}}{5} = \frac{\text{Fahrenheit} - 32}{9}$$

- Write a PHP script that prompts the user to enter an integer, the type of operation (+, -, *, /, DIV, MOD, POWER), and a second integer. The script should execute the required operation and display the result.
- Design a flowchart and write the corresponding PHP script that finds and displays the value of y (if possible) in the following formula.

$$y = \frac{5x + 3}{x - 5} + \frac{3x^2 + 2x + 2}{x + 1}$$

- Design a flowchart and write the corresponding PHP script that finds and displays the values of y (if possible) in the following formula.

$$y = \begin{cases} \frac{x^2}{x+1} + \frac{3-\sqrt{x}}{x+2}, & x \geq 10 \\ \frac{40x}{x-9} + 3x, & x < 10 \end{cases}$$

Moreover, an error message should be displayed when the user enters any non-numeric characters.

7. Write a PHP script that finds and displays the values of y (if possible) in the following formula.

$$y = \left\{ \begin{array}{ll} \frac{x}{\sqrt{x+30}} + \frac{(8+x)^2}{x+1}, & -15 < x \leq -10 \\ \frac{|40x|}{x-8}, & -10 < x \leq 0 \\ \frac{3x}{\sqrt{x-9}}, & 0 < x \leq 25 \\ x-1, & \text{for all other values of } x \end{array} \right\}$$

8. Write a PHP script that prompts the user to enter the ages of three people and then finds and displays the age in the middle.
9. Write a PHP script that prompts the user to enter the name and the ages of three people and then displays the name of the youngest person or the oldest person, depending on which one is closer to the third age in the middle.
10. A three-digit integer is called an Armstrong number when the sum of the cubes of its digits is equal to the number itself. The number 371 is such a number, since $3^3 + 7^3 + 1^3 = 371$. Write a PHP script that lets the user enter a three-digit integer and then displays a message indicating whether or not the given number is an Armstrong one. Moreover, individual error messages should be displayed when the user enters any non-numeric characters or any numbers other than three-digit ones.
11. Write a PHP script that prompts the user to enter a day (1 – 31), a month (1 – 12), and a year and then finds and displays how many days are left until the end of that month. The script should take into consideration the leap years. In the case of a leap year, February has 29 instead of 28 days.
12. Write a PHP script that lets the user enter a word of six letters and then displays a message indicating whether or not every second letter is capitalized. The word “AtHeNa” is such a word, but it can be also given as “aThEnA”.
13. A book store sells e-books for \$10 each. Quantity discounts are given according to the following table.

Quantity	Discount
3 – 5	10%
6 – 9	15%
10 – 13	20%
14 – 19	27%
20 or more	30%

Write a PHP script that prompts the user to enter the total number of e-books purchased and then displays the amount of discount (if any), and the total amount of the purchase after the discount.

14. In a supermarket, the discount that customers receive based on the before-tax amount of their order is presented in next table.

Range	Discount
amount < \$50	0%
\$50 ≤ amount < \$100	1%
\$100 ≤ amount < \$200	2%
\$250 ≤ amount	3%

Write a PHP script that calculates the discount that customers receive (if any) based on the amount of their order after adding a VAT (Value Added Tax) of 19%. Moreover, individual error messages should be displayed when the user enters any non-numeric or negative values.

15. The Body Mass Index (BMI) is often used to determine whether an adult person is overweight or underweight for his or her height. The formula used to calculate the BMI of an adult person is

$$BMI = \frac{weight \cdot 703}{height^2}$$

Write a PHP script that prompts the user to enter his or her age, weight (in pounds) and height (in inches) and then displays a description according to the following table.

Body Mass Index	Description
BMI < 15	Very severely underweight
15.0 ≤ BMI < 16.0	Severely underweight
16.0 ≤ BMI < 18.5	Underweight
18.5 ≤ BMI < 25	Normal
25.0 ≤ BMI < 30.0	Overweight
30.0 ≤ BMI < 35.0	Severely overweight
35.0 ≤ BMI	Very severely overweight

The message “Invalid age” should be displayed when the user enters an age less than 18.

16. The LAV Water Company charges for subscribers’ water consumption according to the following table (monthly rates for domestic accounts).

Water Consumption (cubic feet)	USD per cubic foot
consumption ≤ 10	\$3
11 ≤ consumption ≤ 20	\$5
21 ≤ consumption ≤ 35	\$7
36 ≤ consumption	\$9

Write a PHP script that prompts the user to enter the total amount of water consumed (in cubic feet) and then calculates and displays the total amount to pay. Please note that the rates are progressive. Federal, state, and local taxes add a total of 10% to each bill. Moreover, individual error messages should be displayed when the user enters any non-numeric or negative values.

17. Write a PHP script that prompts the user to enter his or her taxable income and the number of his or her children and then calculates the total tax to pay according to the following table. However, total tax is reduced by 2% when the user has at least one child. Please note that the rates are progressive.

Taxable Income (USD)	Tax Rate
income ≤ \$8000	10%
\$8000 < income ≤ \$30000	15%
\$30000 < income ≤ \$70000	25%
\$70000 < income	30%

18. The Beaufort scale is an empirical measure that relates wind speed to observed conditions on land or at sea. Write a PHP script that prompts the user to enter the wind speed and then displays the corresponding Beaufort number and description according to the following table. Moreover, individual error messages should be displayed when the user enters any non-numeric or negative values. Also, if wind speed is 3 Beaufort or less, an additional message “It’s Fishing Day!!!” should be displayed.

Wind Speed (miles per hour)	Beaufort Number	Description
wind speed < 1	0	Calm
1 ≤ wind speed < 4	1	Light air
4 ≤ wind speed < 8	2	Light breeze
8 ≤ wind speed < 13	3	Gentle breeze
13 ≤ wind speed < 18	4	Moderate breeze

$18 \leq \text{wind speed} < 25$	5	Fresh breeze
$25 \leq \text{wind speed} < 31$	6	Strong breeze
$31 \leq \text{wind speed} < 39$	7	Moderate gale
$39 \leq \text{wind speed} < 47$	8	Gale
$47 \leq \text{wind speed} < 55$	9	Strong gale
$55 \leq \text{wind speed} < 64$	10	Storm
$64 \leq \text{wind speed} < 74$	11	Violent storm
$74 \leq \text{wind speed}$	12	Hurricane force

Review Questions in “Decision Control Structures”

Answer the following questions.

1. What is a Boolean expression?
2. Which comparison operators does PHP support?
3. Which logical operator performs a logical conjunction?
4. Which logical operator performs a logical disjunction?
5. When the result of a logical operator AND (&&) is true?
6. When the result of a logical operator OR (||) is true?
7. State the order of precedence of logical operators.
8. State the order of precedence of arithmetic, comparison, and logical operators.
9. Design the flowchart and write the corresponding PHP statement (in general form) of a single-alternative decision structure. Describe how this decision structure operates.
10. Design the flowchart and write the corresponding PHP statement (in general form) of a dual-alternative decision structure. Describe how this decision structure operates.
11. Design the flowchart and write the corresponding PHP statement (in general form) of a multiple-alternative decision structure. Describe how this decision structure operates.
12. Design the flowchart and write the corresponding PHP statement (in general form) of a case decision structure. Describe how this decision structure operates.
13. What does the term “nesting a decision structure” mean?
14. How deep can the nesting of decision control structures go? Is there any practical limit?
15. When is the conversion from a multiple-alternative decision structure to a case decision structure possible?
16. When is the conversion from a nested decision control structure to a multiple-alternative decision structure possible?
17. When is the conversion from a nested decision control structure to a case decision structure possible?
18. What is code indentation?
19. Create a diagram that shows all possible paths for solving a linear equation.
20. Create a diagram that shows all possible paths for solving a quadratic equation.
21. When is a year considered a leap year?
22. What is a palindrome number?

Section 5

Loop Control Structures

Chapter 24

Introduction to Loop Control Structures

24.1 What is a Loop Control Structure?

A loop control structure is a control structure that allows the execution of a statement or block of statements multiple times until a specified condition is met.

24.2 From Sequence Control to Loop Control Structures

The next example lets the user enter four numbers and it then calculates and displays their sum. As you can see, there is no loop control structure yet, just the familiar sequence control structure.

```
<?php
    $a = trim(fgets(STDIN));
    $b = trim(fgets(STDIN));
    $c = trim(fgets(STDIN));
    $d = trim(fgets(STDIN));

    $sum = $a + $b + $c + $d;

    echo $sum;

?>
```

This script is quite short—very short actually! However, think of an analogous script that lets the user enter 1000 numbers instead of four! Can you imagine copying and pasting the statement `trim(fgets(STDIN))` 1000 times? Wouldn't it be much easier if you could write this statement only once but “tell” the computer to execute it 1000 times? Of course it would be! But for this you need a loop control structure!

However, if you want to use a loop control structure in this example, you need to modify this script a little. To do so, you have a riddle to solve: you are allowed to use only two variables, `$a` and `$sum`. Yes, you heard it! This script should prompt the user to enter four numbers and then it will calculate and display their sum, but it must do it with only two variables! Can you find a way? Hmmm... it's obvious what you are thinking right now: *“The only thing that I can do with two variables is to read one single value in `$a` and then assign that value to `$sum`.”* Your thinking is quite correct, and it is presented here.

```
$a = trim(fgets(STDIN));
$sum = $a;
```

which can equivalently be written as

```
$sum = 0;

$a = trim(fgets(STDIN));
$sum = $sum + $a;
```

And now what? Now, there are three things that you can actually do, and these are: think, think, and of course, think!

The first number has been stored in variable `$sum`, so variable `$a` is now free for further use! Thus, you can reuse variable `$a` to read a second value which will also be accumulated in variable `$sum`, as follows.

```
$sum = 0;

$a = trim(fgets(STDIN));
$sum = $sum + $a;

$a = trim(fgets(STDIN));
$sum = $sum + $a;
```

Notice: Statement `$sum = $sum + $a` accumulates the value of `$a` to `$sum`. For example, if variable `$sum` contains the value 5 and variable `$a` contains the value 3, the statement `$sum = $sum + $a` assigns the value 8 to variable `$sum`.

Since the second number has been accumulated in variable `$sum`, variable `$a` can be re-used! Of course, this process can go on again and again until all four numbers are read and accumulated in variable `$sum`. The final PHP script is as follows. Please note that this script does not use any loop control structure yet!

```
<?php
    $sum = 0;

    $a = trim(fgets(STDIN));
    $sum = $sum + $a;

    $a = trim(fgets(STDIN));
    $sum = $sum + $a;

    $a = trim(fgets(STDIN));
    $sum = $sum + $a;

    $a = trim(fgets(STDIN));
    $sum = $sum + $a;

    echo $sum;

?>
```

Since user must enter 4 numbers, this pair of statements should be written 4 times.

Notice: The main difference between this script and the initial one is that this one has four identical pairs of statements.

Of course, this example can be expanded to read and find the sum of more than four numbers. However, as it becomes clear, you can't write that pair of statements over and over again because soon you will realize how painful this is. Moreover, if you

forget to write at least one pair of statements, it will ultimately lead to incorrect results.

What you really need here is to keep only one pair of those statements but use a loop control structure that executes it four times (or even 1000 times, if you wish). You can use something like the following code fragment.

```
<?php
    $sum = 0;

    execute_these_statements_4_times {
        $a = trim(fgets(STDIN));
        $sum = $sum + $a;
    }

    echo $sum;
?>
```

Obviously there isn't any `execute_these_statements_4_times` statement in PHP. This is for demonstration purposes only but soon enough you will learn everything about all the loop control structures that PHP supports!

24.3 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A loop control structure is a structure that allows the execution of a statement or block of statements multiple times until a specified condition is met.
2. It is possible to use a sequence control structure that prompts the user to enter 1000 numbers and then calculates their sum.
3. The following code fragment

```
$sum = 10;
$a = 0;
$sum = $sum + $a;
```

accumulates the value 10 in variable `$sum`.

4. The following PHP script

```
<?php
    $a = 5;
    $sum = $sum + $a;
    echo $sum;
?>
```

satisfies the property of definiteness.

5. The following two code fragments

```
$a = 5;  
$sum = $a;
```

and

```
$sum = 0;  
$a = 5;  
$sum = $sum + $a;
```

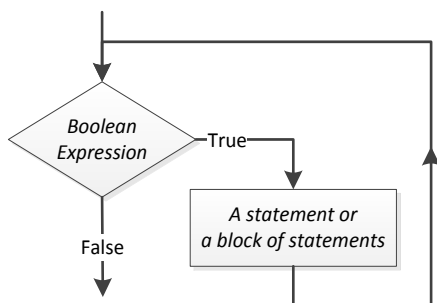
are considered equivalent.

Chapter 25

The Pre-Test Loop Structure

25.1 The Pre-Test Loop Structure

The pre-test loop structure is shown in the following flowchart.



Notice: The Decision symbol (the diamond, or rhombus) is used both in decision control structures and in loop control structures. However, in loop control structures, one of the diamond's exits always has an upward direction.

Let's see what happens when the flow of execution reaches a pre-test loop structure. If the Boolean expression evaluates to `true`, the statement or block of statements of the structure is executed and the flow of execution goes back to the point just above the Decision symbol. If the Boolean expression evaluates to `true` again, the process repeats. When the Boolean expression evaluates to `false`, the flow of execution exits the loop.

Notice: A "pre-test loop structure" is named this way because first the Boolean expression is evaluated, and afterwards the statement or block of statements of the structure is executed.

Notice: Because the Boolean expression is evaluated before entering the loop, a pre-test loop may perform from zero to many iterations.

Notice: Each time the statement or block of statements of the structure is executed, the term used in computer science is "the loop is iterating" or "the loop performs an iteration."

The general form of the PHP statement is

```
while (Boolean_Expression) {  
    A statement or block of statements  
}
```

However, when only one single statement needs to be part of the `while` statement, you are allowed to omit the braces `{ }`. Thus, the `while` statement becomes

```
while (Boolean_Expression)  
    One Single Statement;
```

Remember! *Many programmers prefer to always use braces, even when the `while` statement encloses just one statement.*

The following example displays the numbers 1 to 10.

file_25.1.php

```
<?php
    $i = 1;
    while ($i <= 10) {
        echo $i, "\n";
        $i++;
    }
?>
```

Notice: *Please note that the statements inside the structure are indented. Just as in decision control structures, this indentation makes scripts easier to read and understand. By using indentation, you visually set those statements apart from the rest of the structure.*

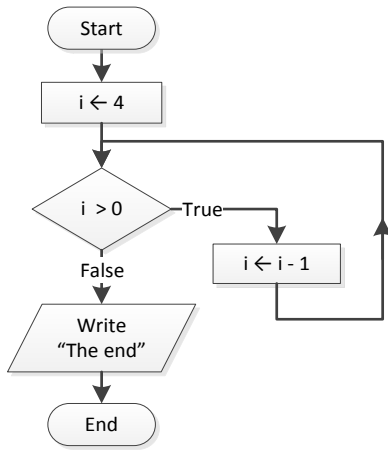
Exercise 25.1-1 *Designing the Flowchart and Counting the Total Number of Iterations*

Design the corresponding flowchart for the following PHP script. How many iterations does this PHP script perform?

```
<?php
    $i = 4;
    while ($i > 0) {
        $i--;
    }
    echo "The end";
?>
```

Solution

The corresponding flowchart is as follows.



Next, a trace table can help you observe the flow of execution.

Step	Statement	Notes	\$i
1	<code>\$i = 4</code>		4
2	<code>while (\$i > 0)</code>	This evaluates to true	3
3	<code>\$i--</code>		
4	<code>while (\$i > 0)</code>	This evaluates to true	2
5	<code>\$i--</code>		
6	<code>while (\$i > 0)</code>	This evaluates to true	1
7	<code>\$i--</code>		
8	<code>while (\$i > 0)</code>	This evaluates to true	0
9	<code>\$i--</code>		
10	<code>while (\$i > 0)</code>	This evaluates to false	
11	<code>echo "The end"</code>	The message "The end" is displayed	

As you can see from the trace table, the total number of iterations is four.

Now, let's draw some conclusions!

- If you want to find the total number of iterations, you need to count the number of times the statement or block of statements of the structure is executed and not the number of times the Boolean expression is evaluated.
- In a pre-test loop structure, when the statement or block of statements of the structure is executed N times, the Boolean expression is evaluated N+1 times.

Exercise 25.1-2 *Counting the Total Number of Iterations*

How many iterations does this PHP script perform?

```
<?php
    $i = 4;
    while ($i >= 0) {
        $i--;
    }
    echo "The end";
?>
```

Solution

This exercise is almost identical to the previous one. The only difference is that the Boolean expression here remains `true`, even for `$i = 0`. Therefore, it performs an additional iteration, that is, five iterations.

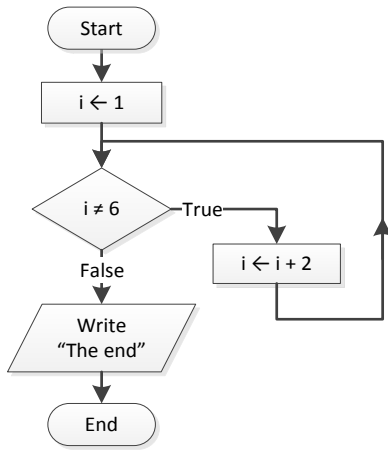
Exercise 25.1-3 *Designing the Flowchart and Counting the Total Number of Iterations*

Design the corresponding flowchart for the following PHP script. How many iterations does this PHP script perform?

```
<?php
    $i = 1;
    while ($i != 6) {
        $i += 2;
    }
    echo "The end";
?>
```

Solution

The corresponding flowchart is as follows.



Now, let’s create a trace table to observe the flow of execution.

Step	Statement	Notes	\$i
1	<code>\$i = 1</code>		1
2	<code>while (\$i != 4)</code>	This evaluates to true	1 st Iteration
3	<code>\$i += 2</code>		3
4	<code>while (\$i != 4)</code>	This evaluates to true	2 nd Iteration
5	<code>\$i += 2</code>		5
6	<code>while (\$i != 4)</code>	This evaluates to true	3 rd Iteration
7	<code>\$i += 2</code>		7
8	<code>while (\$i != 4)</code>	This evaluates to true	.
9	...	...	.
10	...	...	

As you can see from the trace table, since the value 6 is never assigned to variable `$i`, this script will iterate for an infinite number of times! Obviously, this script does not satisfy the property of finiteness.

Exercise 25.1-4 *Counting the Total Number of Iterations*

How many iterations does this PHP script perform?

```
<?php
    $i = -10;
    while ($i > 0) {
        $i--;
    }
    echo "The end";
```

```
?>
```

Solution

Initially, the value -10 is assigned to variable `$i`. The Boolean expression directly evaluates to `false` and the flow of execution goes right to the `echo "The end"` statement. Thus, this script performs zero iterations.

Exercise 25.1-5 Finding the Sum of 10 Numbers

Using a pre-test loop structure, write a PHP script that lets the user enter 10 numbers and then calculates and displays their sum.

Solution

Do you remember the example in Chapter 24 for calculating the sum of four numbers? At the end, after a little work, the proposed PHP script became

```
<?php
    $sum = 0;

    execute_these_statements_4_times {
        $a = trim(fgets(STDIN));
        $sum = $sum + $a;
    }

    echo $sum;
?>
```

Now, you need a way to “present” the statement `execute_these_statements_4_times` with real PHP statements. The `while` statement is actually able to do this, but you need one extra variable to count the total number of iterations. Then, when the desired number of iterations has been performed, the flow of execution should exit the loop.

Following is a general purpose PHP fragment that iterates for the number of times specified by `total_number_of_iterations`,

```
$i = 1;
while ( $i <= total_number_of_iterations ) {
    A statement or block of statements
    $i++;
}
```

where `total_number_of_iterations` can be a constant value or even a variable or an expression.

Notice: The name of the variable `$i` is not binding. You can use any variable name you wish such as `$counter`, `$count`, `$k`, and more.

After combining this script with the previous one, the final script becomes

file_25.1-5.php

```
<?php
    $sum = 0;

    $i = 1;
    while ($i <= 4) {
        $a = trim(fgets(STDIN));
        $sum = $sum + $a;

        $i++;
    }
    echo $sum;
?>
```

This pair of statements is executed 4 times forcing the user to enter 4 numbers.

Exercise 25.1-6 Finding the Product of 20 Numbers

Write a PHP script that lets the user enter 20 numbers and then calculates and displays their product.

Solution

If you were to use a sequence control structure, it would be something like the next code fragment.

```
$p = 1;
```

```
$a = trim(fgets(STDIN));
$p = $p * $a;
```

This should be written 20 times

```
$a = trim(fgets(STDIN));
$p = $p * $a;
```

```
...
```

```
...
```

```
$a = trim(fgets(STDIN));
$p = $p * $a;
```

Notice: Please note that variable `$p` is initialized to 1 instead of 0. This is necessary for the statement `$p = $p * $a` to operate properly; the final product would be zero otherwise.

Using knowledge from the previous exercise, the final script becomes

file_25.1-6.php

```
<?php
    $p = 1;

    $i = 1;
    while ($i <= 20) {
        $a = trim(fgets(STDIN));
        $p = $p * $a;

        $i++;
    }
    echo $p;
?>
```

Exercise 25.1-7 *Finding the Product of N Numbers*

Write a PHP script that lets the user enter N numbers and then calculates and displays their product. The value of N should be given by the user at the beginning of the script.

Solution

In this exercise, the total number of iterations depends on a value that the user must enter. Following is a general purpose PHP fragment that iterates for N times, where N is given by the user.

```
$n = trim(fgets(STDIN));

$i = 1;
while ($i <= $n) {
    A statement or block of statements
    $i++;
}
```

According to what you have learned so far, the final script becomes

file_25.1-7.php

```
<?php
    $n = trim(fgets(STDIN));

    $p = 1;

    $i = 1;
    while ($i <= $n) {
```

```
$a = trim(fgets(STDIN));
$p = $p * $a;

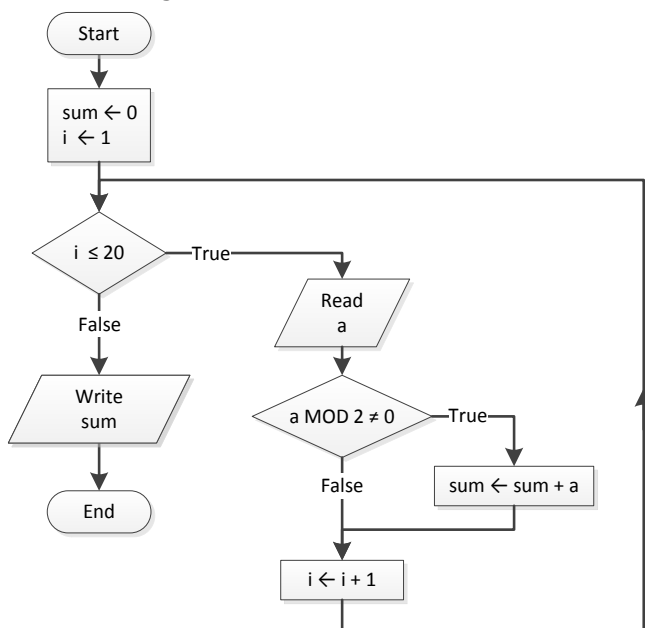
$i++;
}
echo $p;
?>
```

Exercise 25.1-8 Finding the Sum of Odd Numbers

Design a flowchart and write the corresponding PHP script that lets the user enter 20 integers, and then calculates and displays the sum of the odd numbers.

Solution

This is quite easy. What the script must do inside the loop is check whether or not a given number is odd and, if it is, that number should accumulate to variable `$sum`; even numbers should be ignored. The flowchart is as follows.



The corresponding PHP script is as follows.

file_25.1-8.php

```
<?php
    $sum = 0;

    $i = 1;
```



```
while ($i <= 20) {
    $a = trim(fgets(STDIN));
    if ($a % 2 != 0) {
        $sum = $sum + $a;
    }
    $i++;
}
echo $sum;
?>
```

Exercise 25.1-9 Finding the Sum of an Unknown Quantity of Numbers

Write a PHP script that lets the user enter numeric values repeatedly until the value -1 is entered. When data input is completed, the sum of the numbers entered should be displayed. (The value of -1 should not be included in the final sum.) Next, create a trace table to check if your script operates properly using 10, 20, 5, and -1 as input values.

Solution

In this exercise, the total number of iterations is unknown. If you were to use decision control structures, your script would look something like the code fragment that follows.

```
<?php
```

```
    $sum = 0;
```

```
    $a = trim(fgets(STDIN));
```

```
    if ($a != -1) {
```

```
        $sum = $sum + $a;
```

```
        $a = trim(fgets(STDIN));
```

```
    if ($a != -1) {
```

```
        $sum = $sum + $a;
```

```
        $a = trim(fgets(STDIN));
```

```
        if ($a != -1) {
```

```
            $sum = $sum + $a;
```

```
            $a = trim(fgets(STDIN));
```

```
            ...
```

```
            ...
```

```
        }
```

```
    }
```

```
}
```

```
echo $sum;
```

This is the part of the script that actually repeats

?>

Now let’s rewrite this script using a loop control structure instead. The final script is presented next. If you try to follow the flow of execution, you will find that it operates equivalently to the previous one.

file_25.1-9.php

```
<?php
    $sum = 0;

    $a = trim(fgets(STDIN));
    while ($a != -1) {
        $sum = $sum + $a;
        $a = trim(fgets(STDIN));
    }

    echo $sum;
?>
```

Now let’s create a trace table to determine if this script operates properly using 10, 20, 5, and -1 as input values.

Step	Statement	Notes	\$a	\$sum
1	\$sum = 0		?	0
2	\$a = trim(fgets(STDIN))		10	0
3	while (\$a != -1)	This evaluates to true		
4	\$sum = \$sum + \$a		10	10
5	\$a = trim(fgets(STDIN))		20	10
6	while (\$a != -1)	This evaluates to true		
7	\$sum = \$sum + \$a		20	30
8	\$a = trim(fgets(STDIN))		5	30
9	while (\$a != -1)	This evaluates to true		
10	\$sum = \$sum + \$a		5	35
11	\$a = trim(fgets(STDIN))		-1	35
12	while (\$a != -1)	This evaluates to false		
13	echo \$sum	The value 35 is displayed		

As you can see, in the end, variable \$sum contains the value 35, which is, indeed, the sum of the values 10 + 20 + 5. Moreover, the final given value of -1 does not participate in the final sum.

25.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A pre-test loop may perform zero iterations.
2. In flowcharts, both exits of the diamond symbol in a pre-test loop structure, have an upwards direction.
3. The statement or block of statements of a pre-test loop structure is executed at least one time.
4. A pre-test loop stops iterating when its Boolean expression evaluates to **true**.
5. In a pre-test loop structure, when the statement or block of statements of the structure is executed N times, the Boolean expression is evaluated N - 1 times.
6. In the following code fragment

```
$i = 1;
while ($i <= 10)
    echo "Hello";
    $i++;
```

the word "Hello" is displayed 10 times

7. The following PHP script

```
<?php
    $i = 1;
    while ($i != 10) {
        echo "Hello";
        $i += 2;
    }
?>
```

does **not** satisfy the property of finiteness

25.3 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. In flowcharts, the diamond symbol is being used
 - a. in decision control structures.
 - b. in loop control structures.
 - c. all of the above
2. In the following code fragment

```
$i = 1;
while ($i < 10) {
    echo "Hello Hermes";
    $i++;
}
```

the message "Hello Hermes" is displayed

- a. 10 times.
- b. 9 times.
- c. 1 time.
- d. 0 times.
- e. none of the above

3. In the following code fragment

```
$i = 1;  
while ($i < 10)  
    echo "Hi!\n";  
    echo "Hello Ares";  
    $i++;
```

the message "Hello Ares" is displayed

- a. 10 times.
- b. 1 time.
- c. 0 times.
- d. none of the above

4. In the following code fragment

```
$i = 1;  
while ($i < 10)  
    $i++;  
    echo "Hi!\n";  
    echo "Hello Aphrodite";
```

the message "Hello Aphrodite" is displayed

- a. 10 times.
- b. 1 time.
- c. 0 times.
- d. none of the above

5. In the following code fragment

```
$i = 1;  
while ($i >= 10) {  
    echo "Hi!\n";  
    echo "Hello Apollo";  
    $i++;  
}
```

the message "Hello Apollo" is displayed

- a. 10 times.
- b. 1 time.

- c. 0 times.
 - d. none of the above
6. The following PHP script calculates and displays the sum of

```
<?php
    $n = trim(fgets(STDIN));
    $s = 0;
    $i = 1;
    while ($i < $n) {
        $a = trim(fgets(STDIN));
        $s = $s + $a;
        $i++;
    }
    echo $s;
?>
```

- a. as many numbers as the value of variable `$n` denotes.
- b. as many numbers as the result of the expression `$n - 1` denotes.
- c. as many numbers as the value of variable `$i` denotes.
- d. none of the above

25.4 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
    $i = 3;
    $x = 0;
    while ($i >= 0) {
        $i--;
        $x += $i;
    }
    echo $x;
?>
```

2. Design the corresponding flowchart and create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
    $i = -5;
    while ($i > 10) {
        $i--;
    }
```

```
echo $i;  
?>
```

3. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php  
$a = 2;  
while ($a <= 10) {  
    $b = $a + 1;  
    $c = $b * 2;  
    $d = $c - $b + 1;  
    switch ($d) {  
        case 4:  
            echo $b, $c;  
            break;  
        case 5:  
            echo $c;  
            break;  
        case 8:  
            echo $a, $b;  
            break;  
        default:  
            echo $a, $b, $d;  
    }  
    $a += 4;  
}  
?>
```

4. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php  
$a = 1;  
$b = 1;  
$c = 0;  
$d = 0;  
while ($b < 2) {  
    $x = $a + $b;  
    if ($x % 2 != 0)  
        $c = $c + 1;  
    else  
        $d = $d + 1;  
    $a = $b;
```

```
$b = $c;
$c = $d;
}
?>
```

5. Fill in the gaps in the following code fragments so that all loops perform exactly four iterations.

i.

```
$a = 3;
while ($a > ..... ) {
    $a--;
}
```

ii.

```
$a = 5;
while ($a < ..... ) {
    $a++;
}
```

iii.

```
$a = 9;
while ($a != 10 ) {
    $a = $a + ..... ;
}
```

iv.

```
$a = 1;
while ($a != ..... ) {
    $a -= 2;
}
```

v.

```
$a = 2;
while ($a < ..... ) {
    $a = 2 * $a;
}
```

vi.

```
$a = 1;
while ($a < ..... ) {
    $a = $a + 0.1;
}
```

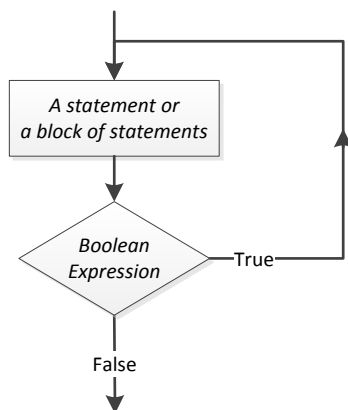
6. Write a PHP script that lets the user enter N numbers and then calculates and displays their sum and their average. The value of N should be given by the user at the beginning of the script.
7. Write a PHP script that lets the user enter N integers and then calculates and displays the product of those that are even. The value of N should be given by the user at the beginning of the script.
8. Write a PHP script that lets the user enter 100 integers and then calculates and displays the sum of those with a last digit of 0. For example, the values 10, 2130, and 500 are such numbers.
Hint: You can isolate the last digit of any integer using a modulus 10 operation.
9. Write a PHP script that lets the user enter 20 integers and then calculates and displays the sum of those that consist of three digits.
Hint: All three-digit integers are between 100 and 999.
10. Write a PHP script that lets the user enter numeric values repeatedly until the value 0 is entered. When data input is completed, the product of the numbers entered should be displayed. (The last 0 entered should not be included in the final product.) Next, create a trace table to check if your script operates properly using 3, 2, 9, and 0 as input values.
11. The population of a town is now at 30000 and is expanding at a rate of 3% per year. Write a PHP script to determine how many years it will take for the population to exceed 100000.

Chapter 26

The Post-Test Loop Structure

26.1 The Post-Test Loop Structure

The post-test loop structure is shown in the following flowchart.



Remember! In loop control structures, one of the diamond's exits always has an upward direction.

Let's see what happens when the flow of execution reaches a post-test loop structure. The statement or block of statements of the structure is directly executed and if the Boolean expression evaluates to `true`, the flow of execution goes back to the point just above the statement or block of statements of the structure. The statement or block of statements is executed once more and if the Boolean expression evaluates to `true` again, the process repeats. When the Boolean expression evaluates to `false`, the flow of execution exits the loop.

Notice: The post-test loop differs from the pre-test loop in that first the statement or block of statements of the structure is executed and afterwards the Boolean expression is evaluated. Consequently, the post-test loop performs at least one iteration!

Notice: Each time the statement or block of statements of the structure is executed, the term used in computer science is "the loop is iterating" or "the loop performs an iteration."

The general form of the PHP statement is

```
do {  
    A statement or block of statements  
} while (Boolean_Expression);
```

However, when only one single statement needs to be part of the `do-while` statement, you are allowed to omit the braces `{ }`. Thus the `do-while` statement becomes

```
do  
    One Single Statement;  
while (Boolean_Expression);
```

Remember! Many programmers prefer to always use braces even when the `do-while` statement encloses just one statement.

The following example displays the numbers 1 to 10.

file_26.1.php

```
<?php
    $i = 1;
    do {
        echo $i, "\n";
        $i++;
    } while ($i <= 10);
?>
```

Notice: Please note the presence of a semicolon (;) character at the end of the `do-while` statement.

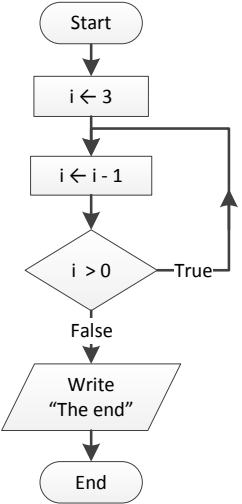
Exercise 26.1-1 *Designing the Flowchart and Counting the Total Number of Iterations*

Design the corresponding flowchart for the following PHP script. How many iterations does this PHP script perform?

```
<?php
    $i = 3;
    do {
        $i--;
    } while ($i > 0);
    echo "The end";
?>
```

Solution

The corresponding flowchart is as follows.



Now, let’s create a trace table to observe the flow of execution.

Step	Statement	Notes	\$i	
1	\$i = 3		3	
2	\$i--		2	} 1 st Iteration
3	while (\$i > 0)	This evaluates to true		
4	\$i--		1	} 2 nd Iteration
5	while (\$i > 0)	This evaluates to true		
6	\$i--		0	} 3 rd Iteration
7	while (\$i > 0)	This evaluates to false		
8	echo "The end"	The message “The end” is displayed		

As you can see from the trace table, the total number of iterations is three.

Now, let’s draw some conclusions!

- If you have to find the total number of iterations, you can count the number of times the statement or block of statements of the structure is executed but also the number of times the Boolean expression is evaluated, since both numbers are equal.
- In a post-test loop structure, when the statement or block of statements of the structure is executed N times, the Boolean expression is evaluated N times as well.

Exercise 26.1-2 *Counting the Total Number of Iterations*

How many iterations does this PHP script perform?

```
<?php
    $i = 3;
    do {
        $i--;
    } while ($i >= 0);
    echo "The end";
?>
```

Solution

This exercise is almost identical to the previous one. The only difference is that the Boolean expression here remains true even for `$i = 0`. Therefore, it performs an additional iteration, that is, four iterations.

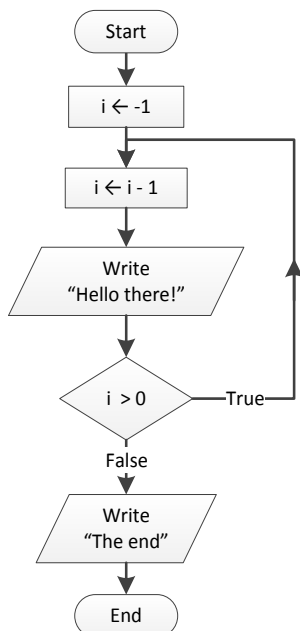
Exercise 26.1-3 *Designing the Flowchart and Counting the Total Number of Iterations*

Design the corresponding flowchart for the following PHP script. How many iterations does this PHP script perform? What happens if you switch the post-test loop structure with a pre-test loop structure?

```
<?php
    $i = -1;
    do {
        $i--;
        echo "Hello there!\n";
    } while ($i > 0);
    echo "The end";
?>
```

Solution

The corresponding flowchart is as follows.



Initially the value -1 is assigned to variable `$i`. Inside the loop, variable `$i` is decremented by one (value -2 is assigned) and the message “Hello there!” is displayed. Finally, the Boolean expression evaluates to `false` and the flow of execution goes right to the `echo "The end"` statement. Thus, this script performs one iteration!

Now, let’s see what happens when a pre-test loop structure is used instead.

```

<?php
    $i = -1;
    while ($i > 0) {
        $i--;
        echo "Hello there!\n";
    }
    echo "The end";
?>
  
```

The value -1 is assigned to variable `$i`. The Boolean expression evaluates to `false` and the flow of execution goes directly to the `echo "The end"` statement. Thus, this script performs zero iterations!

Remember! A pre-test loop structure may perform zero iterations in contrast to the post-test loop structure which performs **at least** one iteration!

Exercise 26.1-4

Counting the Total Number of Iterations

How many iterations does this PHP script perform?

```
<?php
    $i = 1;
    do {
        $i = $i + 2;
    } while ($i != 4);
    echo "The end";
?>
```

Solution

Let’s create a trace table to observe the flow of execution.

Step	Statement	Notes	\$i
1	\$i = 1		1
2	\$i = \$i + 2		3
3	while (\$i != 4)	This evaluates to true	
4	\$i = \$i + 2		5
5	while (\$i != 4)	This evaluates to true	
6	\$i = \$i + 2		7
7	while (\$i != 4)	This evaluates to true	
8	...	...	
9	...	...	

1st Iteration

2nd Iteration

3rd Iteration

.

.

As you can see from the trace table, since the value 4 is never assigned to variable \$i, this script will iterate for an infinite number of times! Obviously, this script does not satisfy the property of finiteness.

Exercise 26.1-5

Finding the Product of N Numbers

Write a PHP script that lets the user enter N numbers and then calculates and displays their product. The value of N should be given by the user at the beginning of the script. What happens if you switch the post-test loop structure with a pre-test loop structure?

Solution

The script that finds and displays the product of N numbers using a pre-test loop structure has already been analyzed in Chapter 25, so, there is no need to explain any details again. For your convenience however, the following solution uses both a pre-test and a post-test loop structure.

file_26.1-5a.php

```
<?php
    $n = trim(fgets(STDIN));

    $p = 1;

    $i = 1;
    while ($i <= $n) {
        $a = trim(fgets(STDIN));
        $p = $p * $a;

        $i++;
    }

    echo $p;
?>
```

file_26.1-5b.php

```
<?php
    $n = trim(fgets(STDIN));

    $p = 1;

    $i = 1;
    do {
        $a = trim(fgets(STDIN));
        $p = $p * $a;

        $i++;
    } while ($i <= $n);

    echo $p;
?>
```

Now watch carefully! If the user enters any value greater than zero, both scripts operate exactly the same way.

The two PHP scripts, however, operate in different ways when the user enters a non-positive¹ value. For example, if the value 0 is entered, the left script performs **zero** iterations whereas the right script performs **one** iteration!

Remember! A pre-test loop structure may perform zero iterations in contrast to the post-test loop structure, which performs **at least** one iteration!

If you want to make both scripts equivalent, you need to add to the right script an `if` statement (just before the `do` keyword) in order to check if variable `$n` contains a positive value. But it's not the right time for this conversation yet. In Chapter 29, you will learn all about converting a pre-test loop structure to its post-test loop structure equivalent and vice versa and unquestionably, after conversion, both scripts (the initial and the resulting one) will be totally equivalent, which means that they will perform an equal number of iterations for any given input values.

Exercise 26.1-6 Finding the Product of an Unknown Quantity of Numbers

Write a PHP script that lets the user enter numeric values repeatedly until the value `-1` is entered. When data input is completed, the product of the numbers entered should be displayed. (The value of `-1` should not be included in final product.) Next, create a trace table to check if your script operates properly using 2, 4, 5, and `-1` as input values.

¹ A quantity that is either zero or negative.

Solution

The PHP script is as follows.

file_26.1-6.php

```
<?php
    $p = 1;

    $a = trim(fgets(STDIN));
    do {
        $p = $p * $a;
        $a = trim(fgets(STDIN));
    } while ($a != -1);

    echo $p;

?>
```

Now let’s create a trace table to determine if the script operates properly.

Step	Statement	Notes	\$a	\$p
1	\$p = 1		?	1
2	\$a = trim(fgets(STDIN))		2	1
3	\$p = \$p * \$a		2	2
4	\$a = trim(fgets(STDIN))		4	2
5	while (\$a != -1)	This evaluates to true		
6	\$p = \$p * \$a		4	8
7	\$a = trim(fgets(STDIN))		5	8
8	while (\$a != -1)	This evaluates to true		
9	\$p = \$p * \$a		5	40
10	\$a = trim(fgets(STDIN))		-1	40
11	while (\$a != -1)	This evaluates to false		
12	echo \$p	The value 40 is displayed		

As you can see, in the end, variable `$p` contains the value 40, which is, indeed, the product of the values $2 \times 4 \times 5$. Moreover, the final given value -1 does not participate in the final sum.

26.2 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A post-test loop may perform zero iterations.

2. A post-test loop stops iterating when its Boolean expression evaluates to false.
3. In a post-test loop structure, when the statement or block of statements of the structure is executed N times, its Boolean expression is evaluated N times as well.
4. You cannot nest a decision control structure inside a post-test loop structure.
5. In the following code fragment

```
$i = 1;
do {
    echo "Hello";
} while ($i >= 10);
```

the word “Hello” is displayed an infinite number of times.

8. The following PHP script

```
<?php
do {
    echo "Hello";
    $i++;
} while ($i <= 10);
?>
```

satisfies the property of definiteness.

9. The following PHP script

```
<?php
$b = trim(fgets(STDIN));
do {
    $a = 1 / ($b - 1);
    $b++;
} while ($b <= 10);
?>
```

does **not** satisfy the property of definiteness.

26.3 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. A post-test loop structure
 - a. performs one iteration more than the pre-test loop structure does.
 - b. always performs the same number of iterations as the pre-test loop structure does.
 - c. none of the above

2. In a post-test loop structure, the statement or block of statements of the structure
- are executed before the loop's Boolean expression is evaluated.
 - are executed after the loop's Boolean expression is evaluated.
 - none of the above
3. In the following code fragment

```
$i = 1;  
do {  
    echo "Hello Poseidon";  
    $i++;  
} while ($i > 5);
```

the message "Hello Poseidon" is displayed

- 5 times.
 - 1 time.
 - 0 times.
 - none of the above
4. In the following code fragment

```
$i = 1;  
do {  
    echo "Hello Athena";  
    $i += 5;  
} while ($i != 50);
```

the message "Hello Athena" is displayed

- at least one time.
 - at least 10 times.
 - an infinite number of times.
 - all of the above
5. In the following code fragment

```
$i = 0;  
do {  
    echo "Hello Apollo";  
} while ($i > 10);
```

the message "Hello Apollo" is displayed

- at least one time.
- an infinite number of times.
- none of the above

26.4 Review Exercises

Complete the following exercises.

1. Identify the syntax error(s) in the following PHP script.

```
<?php
    $i = 3;
    do
        $i--;
    } while ($i > 0)
    echo "The end";
?>
```

2. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
    $y = 5;
    $x = 38;
    do {
        $y *= 2;
        $x++;
        echo $y;
    } while ($y < $x);
?>
```

3. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
    $x = 1;
    do {
        if ($x % 2 == 0) {
            $x++;
        }
        else {
            $x += 3;
        }
        echo $x;
    } while ($x < 12);
?>
```

4. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
    $y = 2;
```

```
$x = 0;
do {
    $y = pow ($y, 2);
    if ($x < 256) {
        $x = $x + $y;
    }
    echo $x, $y;
} while ($y < 65535);
?>
```

5. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
$a = 2;
$b = 4;
$c = 0;
$d = 0;
do {
    $x = $a + $b;
    if ($x % 2 != 0) {
        $c = $c + 5;
    }
    elseif ($d % 2 == 0) {
        $d = $d + 5;
    }
    else {
        $c = $c + 3;
    }
    $a = $b;
    $b = $d;
} while ($c < 11);
?>
```

6. Fill in the gaps in the following code fragments so that all loops perform exactly six iterations.

i.

```
$a = 5;
do {
    $a--;
} while ($a > ..... );
```

ii.

```
$a = 12;
do {
    $a++;
} while ($a < ..... );
```

```
iii. $a = 20;
do {
    $a = $a + ..... ;
} while ($a != 23 );
```

```
v. $a = 2;
do {
    $a = 2 * $a;
} while ($a != ..... );
```

```
iv. $a = 100;
do {
    $a -= 20;
} while ($a != ..... );
```

```
vi. $a = 10;
do {
    $a = $a + 0.25;
} while ($a <= ..... );
```

7. Fill in the gaps in the following code fragments so that all display the value 10 at the end.

```
i. $x = 0;
$y = 0;
do {
    $x++;
    $y += 2;
} while ($x <= ..... );
echo $y;
```

```
ii. $x = 1;
$y = 20;
do {
    $x--;
    $y -= 2.5;
} while ($x >= ..... );
echo $y;
```

```
iii. $x = 3;
$y = 2.5;
do {
    $x--;
    $y *= 2;
} while ($x >= ..... );
echo $y;
```

```
iv. $x = 30;
$y = 101532;
do {
    $x -= .....;
    $y = intval($y / 10);
} while ($x >= 0);
echo $y;
```

8. Design a flowchart and write the corresponding PHP script that lets the user enter 50 integers and then calculates and displays the sum of those that are odd and the sum of those that are even.
9. Write a PHP script that lets the user enter N integers and then calculates and displays the product of those that are negative. The value of N should be given by the user at the beginning of the script, and the final product should always be displayed as a positive value. Assume that the user enters a value greater than 0 for N.
10. Write a PHP script that prompts the user to enter five integers and then calculates and displays the product of all three-digit integers with a first digit of 5. For example, the values 512, 555, and 593 are all such numbers
Hint: All three-digit integers with a first digit of 5 are between 500 and 599.
11. Write a PHP script that prompts the user to enter positive numbers repeatedly until a non-positive one is entered. When data input is completed, the sum of the numbers entered should be displayed. (The last non-positive number entered should not be included in the final sum.) Next,

create a trace table to check if your script operates properly using 5, 2, 3, and 0 as input values.

12. The population of a beehive is now at 50000, but due to environmental reasons it is contracting at a rate of 10% per year. Write a PHP script to determine how many years it will take for the population to be less than 20000.

Chapter 27

Counted Loop Structures

27.1 Counted Loop Structures

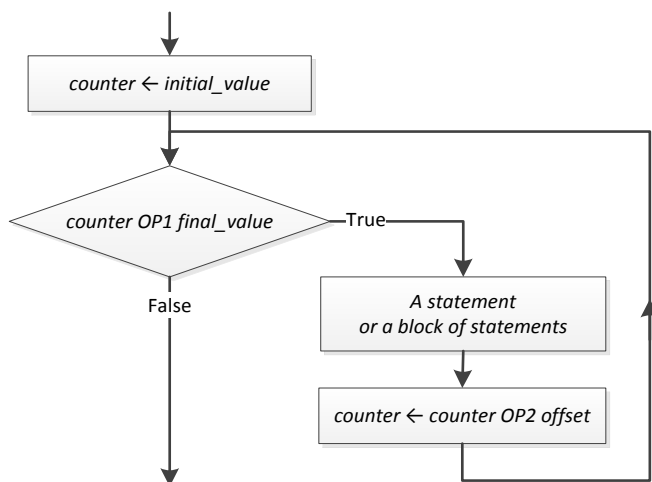
In Chapters 25 and 26, as you certainly noticed, both pre-test and post-test loop structures were used to iterate for a known but also for an unknown number of times. For example, both pre-test and post-test loop structures were used to

- find the sum of 10 given numbers. In this case, the total number of iterations was specific and known even before the loop started iterating.
- find the sum of some given numbers until the user entered the value -1. In this case, the total number of iterations was actually unknown before the loop started iterating.

A counted loop structure, though, can be used only when the number of iterations is known before the loop starts iterating.

The following flowchart is used when iteration for a specific number of times is required. A variable *counter* is used to control the total number of times the loop iterates.

The following flowchart fragment is given in general form.



where

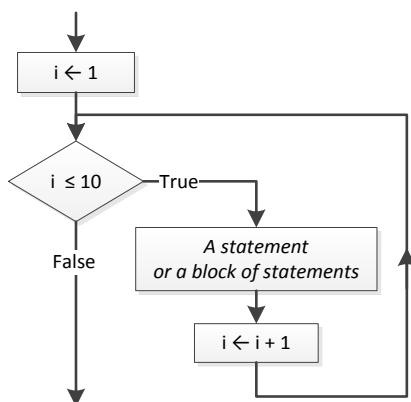
- *counter* must always be a variable
- *initial_value*, *final_value* and *offset* can be a constant value or even a variable or an expression. In case of a variable or an expression, however, the content or the result correspondingly must be invariable inside the loop.
- OP1 must be <= when *counter* increments and >= when *counter* decrements
- OP2 must be + when *counter* increments and – when *counter* decrements

Let's see what happens when the flow of execution reaches a counted loop structure. An initial value is assigned to variable *counter* and if the Boolean expression evaluates to true, the statement or block of statements of the structure is executed. The variable *counter* is incremented (or decremented) by *offset* and the flow of

execution goes back to the point just above the diamond symbol. If the Boolean expression evaluates to `true` again, the process repeats. When the Boolean expression evaluates to `false`, the flow of execution exits the loop.

Notice: A counted loop structure is actually a pre-test loop structure. Because of this, it may perform from zero to many iterations.

Now let's see an example in which variable `i` increments from 1 to 11, allowing the loop to perform 10 iterations.



When the flow of execution reaches the counted loop structure, the initial value 1 is assigned to variable `i`, the Boolean expression evaluates to `true`, and the statement or block of statements of the structure is executed. The variable `i` increments by one and the flow of execution goes back to the point just above the diamond symbol. The Boolean expression evaluates to `true` again and the process repeats. When, after 10 iterations, the value of variable `i` becomes 11, the Boolean expression evaluates to `false` and the flow of execution exits the loop.

Notice: Please note that the flow of execution exits the loop when the value of counter (variable `i`) actually exceeds `final_value`. In this example, when the flow of execution does finally exit the loop, counter `i` does not contain the final value 10 but the next one in order, which is the value 11.

Of course, you can create counted loop structures using the `while` or even the `do-while` statement. However, since counted loop structures are so frequently used in computer programming, almost every computer language, including PHP, incorporates a special statement that is much more readable and convenient than the `while` and `do-while` statements—and this is the `for` statement. Even though the `for` statement in PHP can be written in so many different ways, this book deals only with its basic form, the one that corresponds to the flowchart mentioned before.

In accordance with previous flowchart fragment, the general form of the `for` statement, is

```
for (counter = initial_value; counter OP1 final_value; counter = counter OP2 offset) {
    A statement or block of statements
}
```

Notice: You don't have to increment or decrement the `counter` variable inside the loop because it is automatically done at the end of each loop iteration.

Notice: Don't ever dare alter the value of `counter` inside the loop! The same applies to `initial_value`, `final_value`, and `offset` (in case they are variables and not constant values). This makes your code unreadable and could lead to incorrect results. If you insist, though, please use a `while` or a `do-while` statement instead.

The following example displays the numbers 1 to 10.

file_27.1a.php

```
<?php
for ($i = 1; $i <= 10; $i = $i + 1) {
    echo $i, "\n";
}
?>
```

Of course, when variable `counter` increments (or decrements) by 1, it is more convenient to use incrementing (or decrementing) operators. The previous example can also be written as

file_27.1b.php

```
<?php
for ($i = 1; $i <= 10; $i++) {
    echo $i, "\n";
}
?>
```

The following example displays even numbers from 10 to 2.

file_27.1c.php

```
<?php
for ($i = 10; $i >= 2; $i = $i - 2) {
    echo $i, "\n";
}
?>
```

Notice: Please note that the comparison operator should be `<=` when `counter` increments and `>=` when `counter` decrements.

Of course, when variable `counter` increments or decrements by a value other than 1, it is more convenient to use compound assignment operators. The previous example can also be written as

file_27.1d.php

```
<?php
    for ($i = 10; $i >= 2; $i -= 2) {
        echo $i, "\n";
    }
?>
```

Exercise 27.1-1 Designing the Flowchart and Creating the Trace Table

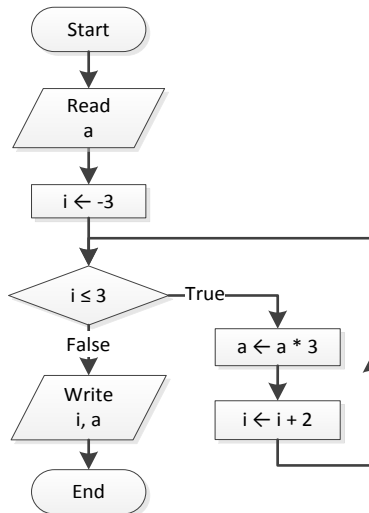
Design the corresponding flowchart and create a trace table to determine the values of the variables in each step of the next PHP script when the input value 1 is entered.

```
<?php
    $a = trim(fgets(STDIN));

    for ($i = -3; $i <= 3; $i += 2) {
        $a = $a * 3;
    }
    echo $i, $a;
?>
```

Solution

The corresponding flowchart is as follows.



Notice: You should keep in mind that a counted loop structure is actually a pre-test loop structure!

If you rewrite the script using the `while` statement the result is as follows.

```
<?php
$a = trim(fgets(STDIN));

$i = -3;
while ($i <= 3) {
    $a = $a * 3;
    $i += 2;
}
echo $i, $a;
?>
```

Now, in order to create a trace table for a `for` statement you have two choices: you can use either the corresponding flowchart or the equivalent script written with the `while` statement.

Step	Statement	Notes	\$a	\$i
1	\$a = trim(fgets(STDIN))		1	?
2	\$i = -3;		1	-3
3	\$i <= 3	This evaluates to true		
4	\$a = \$a * 3		3	-3
5	\$i += 2		3	-1
6	\$i <= 3	This evaluates to true		
7	\$a = \$a * 3		9	-1
8	\$i += 2		9	1
9	\$i <= 3	This evaluates to true		
10	\$a = \$a * 3		27	1
11	\$i += 2		27	3
12	\$i <= 3	This evaluates to true		
13	\$a = \$a * 3		81	3
14	\$i += 2		81	5
15	\$i <= 3	This evaluates to false		
16	echo \$i, \$a	The values 5, 81 are displayed		

Notice: Please note that the flow of execution exits the loop when the value of counter `$i` exceeds `final_value`. In this example, when the flow of execution does finally exit the loop, counter `$i` does not contain the final value 3 but the next one in order, which is the value 5.

Exercise 27.1-2 *Creating the Trace Table*

Create a trace table to determine the values of the variables in each step of the next PHP script when the input value 4 is entered.

```
<?php
    $a = trim(fgets(STDIN));

    for ($i = 6; $i >= $a; $i--) {
        echo $i;
    }
?>
```

Solution

Once again the script can be rewritten using the `while` statement.

```
<?php
    $a = trim(fgets(STDIN));

    $i = 6;
    while ($i >= $a) {
        echo $i;
        $i--;
    }
?>
```

Following is the trace table used to determine the values of the variables in each step.

Step	Statement	Notes	\$a	\$i
1	<code>\$a = trim(fgets(STDIN))</code>		4	?
2	<code>\$i = 6</code>		4	6
3	<code>\$i >= \$a</code>	This evaluates to <code>true</code>		
4	<code>echo \$i</code>	The value 6 is displayed		
5	<code>\$i--</code>		4	5
6	<code>\$i >= \$a</code>	This evaluates to <code>true</code>		
7	<code>echo \$i</code>	The value 5 is displayed		

8	<code>\$i--</code>		4	4
9	<code>\$i >= \$a</code>	This evaluates to true		
10	<code>echo \$i</code>	The value 4 is displayed		
11	<code>\$i--</code>		4	3
12	<code>\$i >= \$a</code>	This evaluates to false		

Exercise 27.1-3 *Counting the Total Number of Iterations*

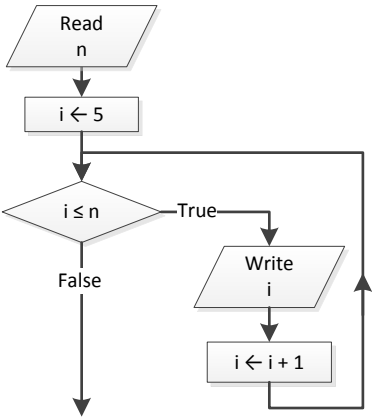
Count the total number of iterations performed by the following code fragment for two different executions.

The input values for the two executions are: (i) 5, and (ii) 6.

```
$n = trim(fgets(STDIN));  
for ($i = 5; $i <= $n; $i++) {  
    echo $i;  
}
```

Solution

In order to better understand what really goes on, instead of creating a trace table, you can just design its corresponding flowchart fragment.



From this flowchart fragment you can see that

- i. for the input value 5, the Boolean expression evaluates to true and the flow of execution enters the loop. Variable `i` increases to 6, the Boolean expression evaluates to false, and the flow of execution exits the loop. Thus, the loop performs one iteration.
- ii. for the input value 6 the loop obviously performs two iterations.

Exercise 27.1-4 Finding the Sum of 10 Numbers

Write a PHP script that prompts the user enter 10 numbers and then calculates and displays their sum.

Solution

In Chapter 25, the solution proposed with a `while` statement was the following:

```
<?php
    $sum = 0;

    $i = 1;
    while ($i <= 10) {
        $a = trim(fgets(STDIN));
        $sum = $sum + $a;

        $i++;
    }
    echo $sum;
?>
```

It's now very easy to rewrite it using a `for` statement and have it display a prompt message before every data input.

file_27.1-4.php

```
<?php
    $sum = 0;

    for ($i = 1; $i <= 10; $i++) {
        echo "Enter a number: ";
        $a = trim(fgets(STDIN));
        $sum = $sum + $a;
    }
    echo $sum;
?>
```

Notice: Please note the absence of the `$i++` statement inside the loop control structure. In a `for` statement, counter (variable `$i`) automatically increases, either way, at the end of each loop iteration.

Exercise 27.1-5 Finding the Square Roots from 0 to N

Write a PHP script that prompts the user to enter an integer and then calculates and displays the square root of all integers from 0 to that given integer.

Solution

In this exercise the counted loop structure should increase the value of variable *counter*, starting from 0, until it reaches the given value. The PHP script is as follows.

file_27.1-5.php

```
<?php
echo "Enter an integer: ";
$num = trim(fgets(STDIN));

for ($i = 0; $i <= $num; $i++) {
    echo sqrt($i), "\n";
}
?>
```

27.2 Rules that Apply to Counted Loop Structures

There are certain rules you should always follow when writing scripts with counted loop structures, since they can save you from undesirable side effects.

- **Rule 1:** The *counter* variable can appear in a statement inside the loop but its value should never be altered. The same applies to *initial_value*, *final_value*, and *offset* in case they are variables and not constant values.
- **Rule 2:** The *offset* should never be zero. If it is set to zero, the loop performs an infinite number of iterations!
- **Rule 3:** If *initial_value* is smaller than *final_value* and the *offset* is negative, the loop performs zero iterations. Breaking this rule on purpose, however, can be useful in certain situations.
- **Rule 4:** If *initial_value* is greater than *final_value* and the *offset* is positive, the loop performs zero iterations. Breaking this rule on purpose, however, can be useful in certain situations.

Exercise 27.2-1 Counting the Total Number of Iterations

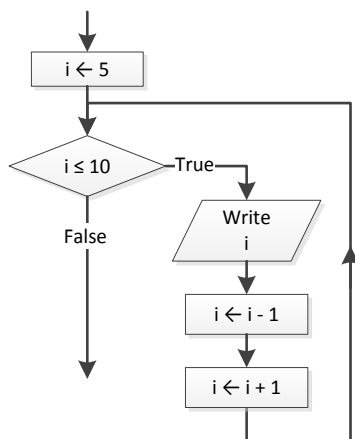
How many iterations does the following code fragment perform?

```
for ($i = 5; $i <= 10; $i++) {
    echo $i;
    $i--;
}
```

Solution

This script breaks the first rule of counted loop structures, which states, *The counter variable can appear in a statement inside the loop but its value should never be altered.*

The corresponding flowchart fragment that follows can help you better understand what really goes on.



As you can see, since the initial value 5 of variable `i` is less than 10, the flow of execution enters the loop. Inside the loop, however, the statement `i <- i - 1` eliminates the statement `i <- i + 1` and this results in a non-incrementing variable `i`, which can never reach *final_value* 10. Thus, the loop performs an infinite number of iterations.

Remember! You should never alter the value of counter variable inside the loop. The same applies to *initial_value*, *final_value*, and *offset* (in case they are variables and not constant values).

Exercise 27.2-2 Counting the Total Number of Iterations

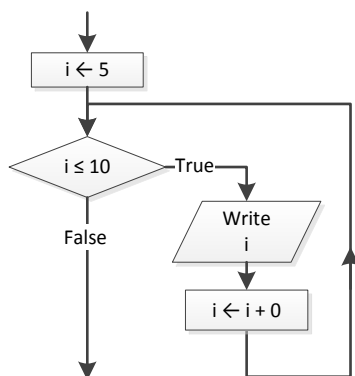
How many iterations does the following code fragment perform?

```
for ($i = 5; $i <= 10; $i += 0) {
    echo $i;
}
```

Solution

This script breaks the second rule of counted loop structures, which states, *The offset should never be zero*.

The corresponding flowchart fragment that follows can help you better understand what really goes on.



As you can see, since the initial value 5 of variable `i` is less than 10, the flow of execution enters the loop. However, the statement `i ← i + 0` never increments variable `i` and this results in a non-incrementing variable `i`, which can never reach *final_value* 10. Thus, the loop performs an infinite number of iterations.

Remember! *The offset should never be zero.*

Exercise 27.2-3 Counting the Total Number of Iterations

How many iterations does the following code fragment perform?

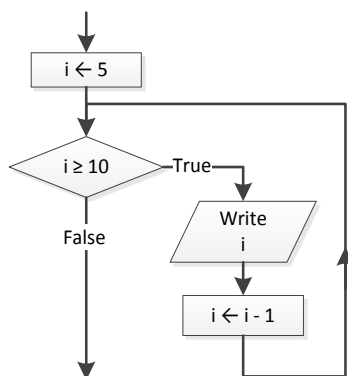
```

for ($i = 5; $i >= 10; $i--) {
    echo $i;
}
  
```

Solution

This script breaks the third rule of counted loop structures, which states, *If initial_value is smaller than final_value and the offset is negative, the loop performs zero iterations.*

The corresponding flowchart fragment that follows can help you better understand what really goes on.



When the flow of execution reaches the loop control structure, the value 5 is assigned to variable `i`. However, the Boolean expression evaluates to `false` and the flow of execution never enters the loop! Thus, the loop performs zero iterations.

Remember! Breaking the third rule of counted loop structures on purpose can be useful in certain situations.

Exercise 27.2-4 Counting the Total Number of Iterations

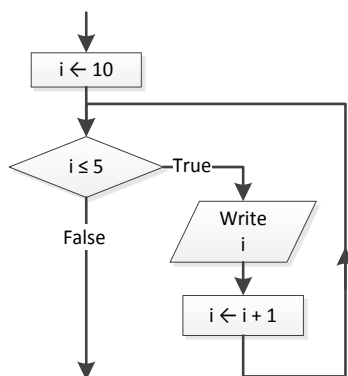
How many iterations does the following code fragment perform?

```
for ($i = 10; $i <= 5; $i++) {
    echo $i;
}
```

Solution

This script breaks the fourth rule of counted loop structures, which states, *If initial_value is greater than final_value and the offset is positive, the loop performs zero iterations.*

The corresponding flowchart fragment that follows can help you better understand what really goes on.



When the flow of execution reaches the loop control structure, the value 10 is assigned to variable *i*. However, the Boolean expression evaluates to *false* and the flow of execution never enters the loop! Thus, the loop performs zero iterations.

Notice: *Breaking the fourth rule of counted loop structures on purpose can be useful in certain situations.*

Exercise 27.2-5 Finding the Average Value of *N* Numbers

Write a PHP script that prompts the user to enter *N* numbers and then calculates and displays their average value. The value of *N* should be given by the user at the beginning of the script. Add all necessary checks to make the script satisfy the property of definiteness.

Solution

The solution is presented next.

file_27.2-5.php

```

<?php
echo "Enter quantity of numbers to enter: ";
$n = trim(fgets(STDIN));

$sum = 0;
for ($i = 1; $i <= $n; $i++) {
    echo "Enter number No ", $i, ": ";
    $a = trim(fgets(STDIN));
    $sum = $sum + $a;
}

if ($n > 0) {
    $average = $sum / $n;
}
  
```

```
    echo "The average value is: ", $average;
}
else {
    echo "You didn't enter any number!";
}
?>
```

Notice: Please note that if the user enters a non-positive value for variable `$n`, the `for` statement performs zero iterations. Even though it breaks the fourth rule of counted loop structures, in this particular exercise, this situation is very useful.

Notice: The `if ($n > 0)` statement, next to the loop control structure, is necessary because if the user enters the value 0 for variable `$n`, the script avoids any division-by-zero errors, thereby satisfying the property of definiteness.

27.3 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. In a counted loop structure, the value of `counter` increments or decrements automatically at the end of each loop.
2. A counted loop structure can be used when the number of iterations is known.
3. In a counted loop structure, the statement or block of statements of the loop is executed at least one time.
4. In a counted loop structure, the `initial_value` cannot be greater than the `final_value`.
5. When flow of execution exits a counted loop structure, the value of `counter` is equal to `final_value`.
6. In a counted loop structure, the value of `initial_value`, `final_value` and `offset` can be either an integer or a float.
7. The only case in which a counted loop structure does not satisfy the property of finiteness is when `offset` is set to zero.
8. In a counted loop structure, the `counter` variable can appear in a statement inside the loop but its value should never be altered.
9. In a counted loop structure, the `offset` can be zero for certain situations.
10. In the following code fragment

```
for ($i = 0; $i <= 10; $i++) {
    echo "Hello";
}
```

the word "Hello" is displayed 10 times.

11. The following code fragment

```
for ($i = 0; $i <= 10; $i--) {
```

```
echo "Hello";  
}
```

satisfies the property of finiteness.

12. The following code fragment

```
$b = trim(fgets(STDIN));  
for ($i = 0; $i <= 10; $i++) {  
    $a = sqrt($b) + $i;  
    $b *= 2;  
}
```

satisfies the property of definiteness.

27.4 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. A counted loop structure
 - a. executes one iteration more than the equivalent post-test loop structure does.
 - b. executes one iteration less than the equivalent post-test loop structure does.
 - c. none of the above
2. A counted loop structure can be used in a problem in which
 - a. the user enters numbers repeatedly until the value -1 is entered.
 - b. the user enters numbers repeatedly until the value entered is greater than *final_value*.
 - c. all of the above
 - d. none of the above
3. In a counted loop structure *initial_value*, *final_value*, and *offset* can be
 - a. a constant value.
 - b. a variable.
 - c. an expression.
 - d. all of the above
4. In a counted loop structure, when *initial_value*, *final_value*, and *offset* are variables, their values
 - a. cannot change inside the loop.
 - b. should not change inside the loop.
 - c. none of the above
5. In a counted loop structure, when *counter* increments, the comparison operator should be
 - a. >=.
 - b. ==.

- c. `<=`.
 - d. none of the above
6. In a counted loop structure, the initial value of *counter*
- a. must be 1.
 - b. can be 1.
 - c. cannot be a negative one.
 - d. none of the above
7. In a counted loop structure, the variable *counter* increments or decrements automatically
- a. at the end of each iteration.
 - b. at the beginning of each iteration.
 - c. It does not increment automatically.
 - d. none of the above
8. In the following code fragment

```
$i = 1;  
for ($i = 5; $i <= 5; $i++) {  
    echo "Hello Hera";  
}
```

the message “Hello Hera” is displayed

- a. 5 times.
 - b. 1 time.
 - c. 0 times.
 - d. none of the above
9. In the following code fragment

```
for ($i = 5; $i <= 4; $i++) {  
    $i = 1;  
    echo "Hello Artemis";  
}
```

the message “Hello Artemis” is displayed

- a. 1 time.
 - b. an infinite number of times.
 - c. 0 times.
 - d. none of the above
10. In the following code fragment

```
for ($i = 5; $i <= 5; $i++) {  
    $i = 1;  
    echo "Hello Ares";  
}
```

the message “Hello Ares” is displayed

- a. 1 time.
- b. an infinite number of times.
- c. 0 times.
- d. none of the above

11. In the following code fragment

```
for ($i = 2; $i <= 8; $i++) {  
    if ($i % 2 == 0) {  
        echo "Hello Demeter";  
    }  
}
```

the message “Hello Demeter” is displayed

- a. 8 times.
- b. 7 times.
- c. 5 times.
- d. none of the above

12. In the following code fragment

```
for ($i = 4; $i <= 5; $i += 0.1) {  
    echo "Hello Dionysus";  
}
```

the message “Hello Dionysus” is displayed

- a. 1 time.
- b. 2 times.
- c. 10 times.
- d. 11 times.

13. In the following code fragment

```
$k = 0;  
for ($i = 1; $i <= 5; $i += 2) {  
    $k = $k + $i;  
}  
echo $i;
```

the value displayed is

- a. 3.
- b. 6.
- c. 9.
- d. none of the above

14. In the following code fragment


```
$k = 0;
for ($i = 100; $i >= -100; $i -= 5) {
    $k = $k + $i;
}
echo $i;
```

the value displayed is

- 95.
- 100.
- 105.
- none of the above

27.5 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the next PHP script. How many iterations does this PHP script perform?

```
<?php
$a = 0;
$b = 0;
for ($j = 0; $j <= 8; $j += 2) {
    if ($j < 5) {
        $b++;
    }
    else {
        $a += $j - 1;
    }
}
echo $a, $b;
?>
```

2. Create a trace table to determine the values of the variables in each step of the next PHP script for two different executions.

The input values for the two executions are: (i) 10, and (ii) 21.

```
<?php
$a = trim(fgets(STDIN));
$b = $a;
for ($j = $a - 5; $j <= $a; $j += 2) {
    if ($j % 2 != 0) {
        $b = $a + $j + 5;
    }
    else {
        $b = $a - $j;
```

```
    }
}
echo $b;
?>
```

3. Create a trace table to determine the values of the variables in each step of the next PHP script for the input value 12.

```
<?php
$a = trim(fgets(STDIN));
for ($j = 2; $j <= $a - 1; $j += 3) {
    $x = $j * 3 + 3;
    $y = $j * 2 + 10;
    if ($y - $x > 0 || $x > 30) {
        $y *= 2;
    }
    $x += 4;
    echo $x, $y;
}
?>
```

4. Fill in the gaps in the following code fragments so that all loops perform exactly five iterations.

i.

```
for ($a = 5; $a <= ..... ; $a++) {
    $b++;
}
```

ii.

```
for ($a = 0; $a <= ..... ; $a += 0.5) {
    $b++;
}
```

iii.

```
for ($a = ..... ; $a >= -15; $a -= 2) {
    $b++;
}
```

iv.

```
for ($a = -11 ; $a >= -15; $a = $a ..... ) {
    $b++;
}
```

5. Design a flowchart and write the corresponding PHP script that prompts the user to enter 20 numbers and then calculates and displays their product and their average value.
6. Write a PHP script that prompts the user to enter a number in degrees and then calculates and displays the sinus of all numbers from 0 to that given number, using an offset of 0.5. It is given that $2\pi = 360^\circ$.

7. Design a flowchart and write the corresponding PHP script that prompts the user to enter 30 four-digit integers and then calculates and displays the sum of those with a first digit of 5 and a last digit of 3. For example, values 5003, 5923, and 5553 are all such integers.
8. Design a flowchart and write the corresponding PHP script that prompts the user to enter N integers and then displays the total number of those that are even. The value of N should be given by the user at the beginning of the script. Moreover, if all integers given are odd, the message “You entered no even integers” should be displayed.
9. Design a flowchart and write the corresponding PHP script that prompts the user to enter 50 integers and then calculates and displays the average value of those that are odd and the average value of those that are even.
10. Design a flowchart and write the corresponding PHP script that prompts the user to enter two integers into variables `$start` and `$finish` and then displays all integers from `$start` to `$finish`. However, at the beginning the script should check if variable `$start` is bigger than variable `$finish`. If this happens, the script should swap their values so that they are always in the proper order.
11. Design a flowchart and write the corresponding PHP script that prompts the user to enter two integers into variables `$start` and `$finish` and then displays all integers from `$start` to `$finish` that are multiples of five. However, at the beginning the script should check if variable `$start` is bigger than variable `$finish`. If this happens, the script should swap their values so that they are always in the proper order.
12. Write a PHP script that prompts the user to enter a real and an integer and then displays the result of the first number raised to the power of the second number, without using the `pow()` function.
13. Write a PHP script that prompts the user to enter a message and then displays the number of words it contains. For example, if the string entered is “My name is Bill Bouras”, the script should display “The message entered contains 5 words”. Assume that the words are separated by a single space character.

Hint: Use the `strlen()` function to get the number of characters that the given message contains.
14. Write a PHP script that prompts the user to enter any string and then displays the average number of letters in each word. For example, if the string entered is “My name is Aphrodite Boura”, the script should display “The average number of letters in each word is 4.4”. Space characters should not be counted.

Chapter 28

Nested Loop Control Structures

28.1 What is a Nested Loop?

A nested loop is a loop within another loop or, in other words, an inner loop within an outer one.

The outer loop controls the number of complete iterations of the inner loop, which means that the first iteration of the outer loop triggers the inner loop to start iterating until completion. Then the second iteration of the outer loop triggers the inner loop to start iterating until completion again. This process repeats until the outer loop has performed all of its iterations.

Take the following PHP script, for example.

file_28.1.php

```
for ($i = 1; $i <= 2; $i++) {  
    for ($j = 1; $j <= 3; $j++) {  
        echo $i, " ", $j, "\n";  
    }  
}
```

Nested loop

The outer loop, the one that is controlled by variable `$i`, controls the number of complete iterations that the inner loop performs. That is, when variable `$i` contains the value 1, the inner loop performs three iterations (for `$j = 1`, `$j = 2`, and `$j = 3`). The inner loop finishes but the outer loop needs to perform one more iteration (for `$i = 2`), so the inner loop starts over and performs three new iterations again (for `$j = 1`, `$j = 2` and `$j = 3`).

The previous example can be likened to the following one.

```
$i = 1; //outer loop assigns value 1 to variable $i  
for ($j = 1; $j <= 3; $j++) { //and inner loop performs three iterations  
    echo $i, " ", $j, "\n";  
}  
  
$i = 2; //outer loop assigns value 2 to variable $i  
for ($j = 1; $j <= 3; $j++) { //and inner loop starts over and  
    echo $i, " ", $j, "\n"; //performs three new iterations  
}
```

The output result is as follows.



Remember! As long as the syntax rules are not violated, you can nest as many loop control structures as you wish. For practical reasons however, as you move to four or five levels of nesting, the entire loop structure becomes very complex and difficult to understand. However, experience shows that the maximum levels of nesting that you will do in your entire life as a programmer is probably three or four.

Notice: There is no need for the inner and outer loops to be of the same type. For example, a `for` statement may nest (enclose) a `while` statement, or vice versa.

Exercise 28.1-1 *Say “Hello Zeus”. Designing the Flowchart and Counting the Total Number of Iterations*

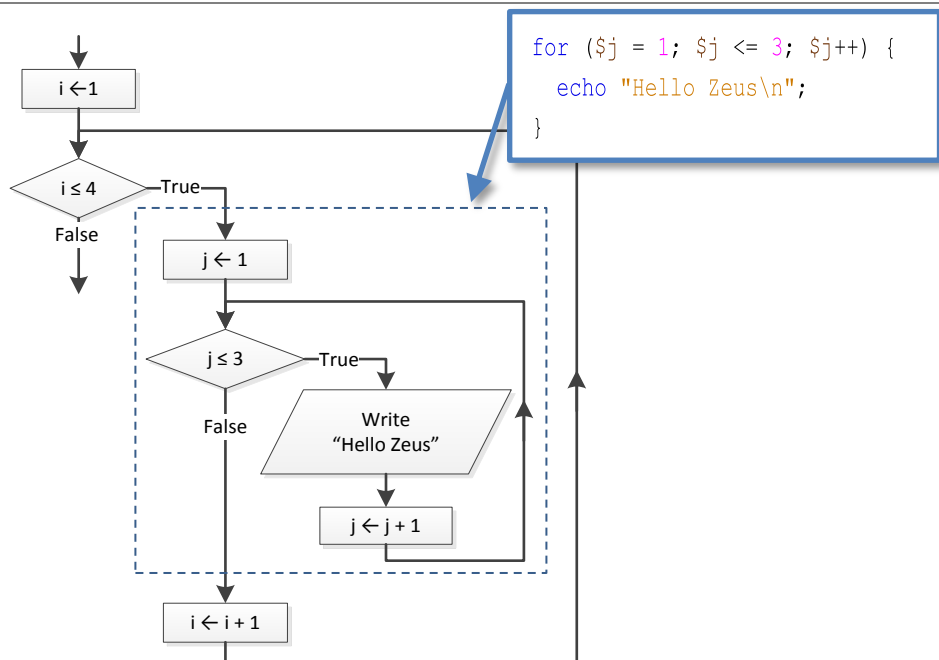
Design the corresponding flowchart for the next code fragment, and find the number of times message “Hello Zeus” is displayed.

file_28.1-1.php

```
for ($i = 1; $i <= 4; $i++) {  
    for ($j = 1; $j <= 3; $j++) {  
        echo "Hello Zeus\n";  
    }  
}
```

Solution

The corresponding flowchart is as follows. The inner loop is marked with a dashed rectangle.



Now, let's try to count the number of times message "Hello Zeus" is displayed. By studying the corresponding flowchart you can see that the values of variables *i* and *j* (in order of appearance) are as follows:

- For *i* = 1, the inner loop performs 3 iterations (for *j* = 1, *j* = 2, and *j* = 3) and the message "Hello Zeus" is displayed 3 times.
- For *i* = 2, the inner loop performs 3 iterations (for *j* = 1, *j* = 2, and *j* = 3) and the message "Hello Zeus" is displayed 3 times.
- For *i* = 3, the inner loop performs 3 iterations (for *j* = 1, *j* = 2, and *j* = 3) and the message "Hello Zeus" is displayed 3 times.
- For *i* = 4, the inner loop performs 3 iterations (for *j* = 1, *j* = 2, and *j* = 3) and the message "Hello Zeus" is displayed 3 times.

Therefore, the message "Hello Zeus" is displayed a total of $4 \times 3 = 12$ times.

Remember! The outer loop controls the number of complete iterations of the inner one!

Exercise 28.1-2 Creating the Trace Table

Design the corresponding flowchart for the next code fragment, and determine the value that variable *\$a* contains at the end.

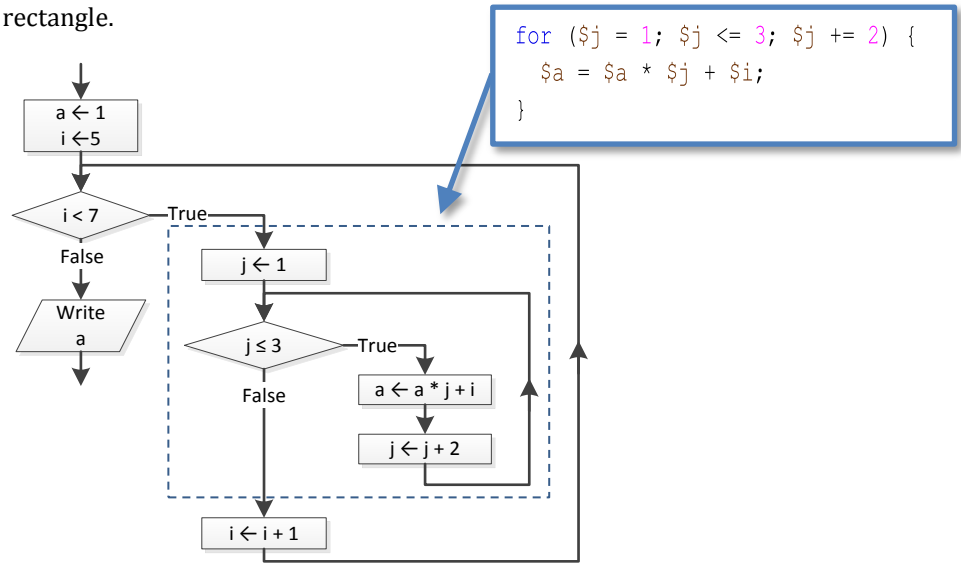
```

$a = 1;
$i = 5;
while ($i < 7) {
    
```

```
for ($j = 1; $j <= 3; $j += 2) {  
    $a = $a * $j + $i;  
}  
$i++;  
}  
echo $a;
```

Solution

Following is the corresponding flowchart. The inner loop is marked with a dashed rectangle.



With the help of the flowchart you can now create the following trace table.

Step	Statement	Notes	\$a	\$i	\$j
1	\$a = 1		1	?	?
2	\$i = 5		1	5	?
3	\$i < 7	This evaluates to true			
4	\$j = 1		1	5	1
5	\$j <= 3	This evaluates to true			
6	\$a = \$a * \$j + \$i		6	5	1
7	\$j = \$j + 2		6	5	3
8	\$j <= 3	This evaluates to true			
9	\$a = \$a * \$j + \$i		23	5	3

10	<code>\$j = \$j + 2</code>		23	5	5
11	<code>\$j <= 3</code>	This evaluates to false			
12	<code>\$i = \$i + 1</code>		23	6	5
13	<code>\$i < 7</code>	This evaluates to true			
14	<code>\$j = 1</code>		23	6	1
15	<code>\$j <= 3</code>	This evaluates to true			
16	<code>\$a = \$a * \$j + \$i</code>		29	6	1
17	<code>\$j = \$j + 2</code>		29	6	3
18	<code>\$j <= 3</code>	This evaluates to true			
19	<code>\$a = \$a * \$j + \$i</code>		93	6	3
20	<code>\$j = \$j + 2</code>		93	6	5
21	<code>\$j <= 3</code>	This evaluates to false			
22	<code>\$i = \$i + 1</code>		93	7	5
23	<code>\$i < 7</code>	This evaluates to false			
24	<code>echo \$a</code>	The value 93 is displayed			

At the end of the script, variable `$a` contains the value 93.

28.2 Rules that Apply to Nested Loops

Beyond the four rules that apply to counted loop structures, there are two extra rules that you should always follow when writing scripts with nested loops, since they can save you from undesirable side effects.

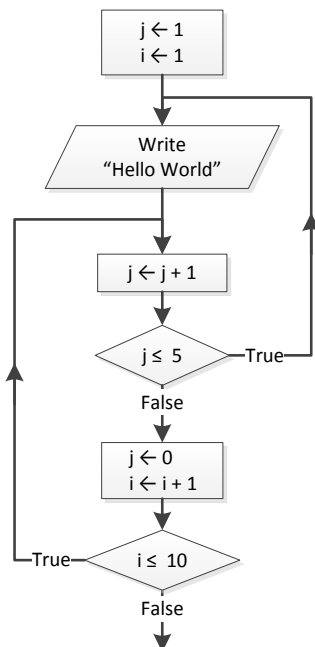
- **Rule 1:** The inner loop must begin and end entirely within the outer loop, which means that the loops must not overlap.
- **Rule 2:** An outer loop and the inner (nested) loop must not use the same *counter* variable.

Exercise 28.2-1 Breaking the First Rule

Design a flowchart fragment that breaks the first rule of nested loops, which states, “The inner loop must begin and end entirely within the outer loop.”

Solution

The following flowchart fragment breaks the first rule of nested loops.



If you try to follow the flow of execution, you can see how smoothly it performs $5 \times 10 = 50$ iterations. No one can tell that this flowchart is wrong; on the contrary, it is absolutely correct. The problem, however, is that this flowchart is completely unreadable. No one can tell, at first glance, what this flowchart really does. Moreover, this structure matches none of the already known loop control structures that you have been taught, so it cannot become a PHP script as is. Try to avoid this kind of nested loop!

Exercise 28.2-2 Counting the Total Number of Iterations

Find the number of times message “Hello” is displayed.

```

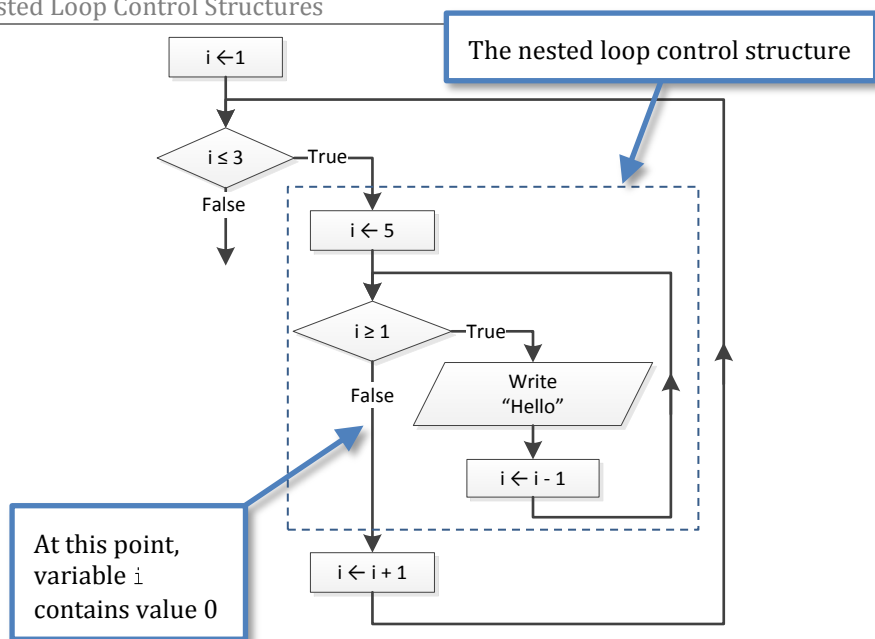
for ($i = 1; $i <= 3; $i++) {
    for ($i = 5; $i >= 1; $i--) {
        echo "Hello";
    }
}

```

Solution

This script breaks the second rule of nested loops, which states, “An outer loop and the inner (nested) loop must not use the same counter variable.”

At first glance, one would think that the word “Hello” is displayed $3 \times 5 = 15$ times. However, a closer second look reveals that things are not always as they seem. Let’s design the corresponding flowchart.



If you try to follow the flow of execution in this flowchart fragment, you can see that when the inner loop completes all of its four iterations, variable *i* contains the value 0. Then, variable *i* increments by 1 and the outer loop repeats again. This process can continue forever since variable *i* can never exceed the value 3 that the Boolean expression of the outer loop requires. Therefore, the message “Hello” is displayed an infinite number of times.

28.3 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. A nested loop is an inner loop within an outer one.
2. The maximum number of levels of nesting in a loop control structure is four.
3. When two loop control structures are nested one within the other, the loop that starts last should complete first.
4. When two loop control structures are nested one within the other, they must not use the same variable as *counter*.
5. In the following code fragment

```

for ($i = 1; $i <= 3; $i++) {
    for ($j = 1; $j <= 3; $j++) {
        echo "Hello";
    }
}

```

the word “Hello” is displayed six times.

6. In the following code fragment

```

for ($i = 1; $i <= 2; $i++) {

```

```
for ($j = 1; $j <= 3; $j++) {  
    for ($k = 1; $k <= 4; $k += 2) {  
        echo "Hello";  
    }  
}
```

the word “Hello” is displayed 12 times.

7. In the following code fragment

```
for ($i = 1; $i <= 3; $i++) {  
    for ($i = 3; $i >= 1; $i--) {  
        echo "Hello";  
    }  
}
```

the word “Hello” is displayed an infinite number of times.

8. In the following code fragment

```
for ($i = 1; $i <= 3; $i++) {  
    $j = 1;  
    do {  
        echo "Hello";  
        $j++;  
    } while ($j < 4);  
}
```

the word “Hello” is displayed nine times.

28.4 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. In the following code fragment

```
for ($i = 1; $i <= 2; $i++) {  
    for ($j = 1; $j <= 2; $j++) {  
        echo "Hello";  
    }  
}
```

the values of variables `$i` and `$j` (in order of appearance) are

- `$j = 1, $i = 1, $j = 1, $i = 2, $j = 2, $i = 1, $j = 2, $i = 2`
- `$i = 1, $j = 1, $i = 1, $j = 2, $i = 2, $j = 1, $i = 2, $j = 2`
- `$i = 1, $j = 1, $i = 2, $j = 2`
- `$j = 1, $i = 1, $j = 2, $i = 2`

2. In the following code fragment

```
$x = 2;
while ($x > -2) {
    do {
        $x--;
        echo "Hello Hestia";
    } while ($x < -2);
}
```

the message "Hello Hestia" is displayed

- 4 times.
- an infinite number of times.
- 0 times.
- none of the above

3. In the following code fragment

```
$x = 1;
while ($x != 500) {
    for ($i = $x; $i <= 3; $i++) {
        echo "Hello Artemis";
    }
    $x++;
}
```

the message "Hello Artemis" is displayed

- an infinite number of times.
- 1500 times.
- 6 times.
- none of the above

4. The following code fragment

```
for ($i = 1; $i <= 3; $i++) {
    for ($j = 1; $j <= $i; $j++) {
        echo $i * $j, ", ";
    }
}
echo "The End!";
```

displays

- 1, 2, 4, 3, 6, 9, The End!
- 1, 2, 3, 4, 6, 9, The End!
- 1, 2, The End!, 4, 3, The End!, 6, 9, The End!
- none of the above

5. The following code fragment

```
for ($i = 1; $i <= 10 ; $i++) {
    for ($i = 10; $i >= 1 ; $i--) {
        echo "Hello Dionysus";
    }
}
```

does **not** satisfy the property of

- definiteness.
- finiteness.
- effectiveness.
- none of the above

28.5 Review Exercises

Complete the following exercises.

- Fill in the gaps in the following code fragments so that all code fragments display the message “Hello Hephaestus” exactly 100 times.

i.

```
for ($a = 7; $a <= ..... ; $a++) {
    for ($b = 1; $b <= 25 ; $b++) {
        echo "Hello Hephaestus\n";
    }
}
```

ii.

```
for ($a = 0; $a <= ..... ; $a += 0.5) {
    for ($b = 10; $b <= 19 ; $b++) {
        echo "Hello Hephaestus\n";
    }
}
```

iii.

```
for ($a = ..... ; $a >= -15; $a -= 2) {
    for ($b = 10; $b >= 0.5 ; $b -= 0.5) {
        echo "Hello Hephaestus\n";
    }
}
```

iv.

```
for ($a = -11 ; $a >= -15; $a -= 1) {
    for ($b = 100; $b <= ..... ; $b += 2) {
        echo "Hello Hephaestus\n";
    }
}
```

- Design the corresponding flowchart and create a trace table to determine the values of the variables in each step of the next code fragment.

```
$a = 1;
```

```

for ($j = 1; $j <= 2; $j += 0.5) {
    $i = 10;
    while ($i < 30) {
        $a = $a + $j + $i;
        $i += 10;
    }
}
echo $a;

```

3. Create a trace table to determine the values of the variables in each step of the next code fragment. How many times is the statement `$s = $s + $i * $j` executed?

```

$s = 0;
for ($i = 1; $i <= 4; $i++) {
    for ($j = 3; $j >= $i; $j--) {
        $s = $s + $i * $j;
    }
}
echo $s;

```

4. Create a trace table to determine the values of the variables in each step of the next PHP script for three different executions. How many iterations does this PHP script perform?

The input values for the three executions are: (i) NO, (ii) YES, NO; and (iii) YES, YES, NO.

```

<?php
    $s = 1;
    $y = 25;
    do {
        for ($i = 1; $i <= 3; $i++) {
            $s = $s + $y;
            $y -= 5;
        }
        $ans = trim(fgets(STDIN));
    } while ($ans == "YES");
    echo $s;
?>

```

5. Write a PHP script that displays an hours and minutes table in the following form.

0	0
0	1
0	2

```

0          3
...
0          59
1          0
1          1
1          2
1          3
...
23         59

```

Please note that the output is aligned with tabs.

6. Using nested loop control structures, write a PHP script that displays the following output.

```

5 5 5 5 5
4 4 4 4
3 3 3
2 2
1

```

7. Using nested loop control structures, write a PHP script that displays the following output.

```

0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5

```

8. Using nested loop control structures, write a PHP script that displays the following rectangle.

```

* * * * *
* * * * *
* * * * *
* * * * *

```

9. Write a PHP script that prompts the user to enter an integer N between 3 and 20 and then displays a square of size N on each side. For example, if the user enters 4 for N, the script should display as shown here.

```

* * * *
* * * *
* * * *
* * * *

```


10. Write a PHP script that prompts the user to enter an integer N between 3 and 20 and then displays a hollow square of size N on each side. For example, if the user enters 4 for N, the script should display as shown here.

```
* * * *
*       *
*       *
* * * *
```

11. Using nested loop control structures, write a PHP script that displays the following triangle.

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

Chapter 29

Tips and Tricks with Loop Control Structures

29.1 Introduction

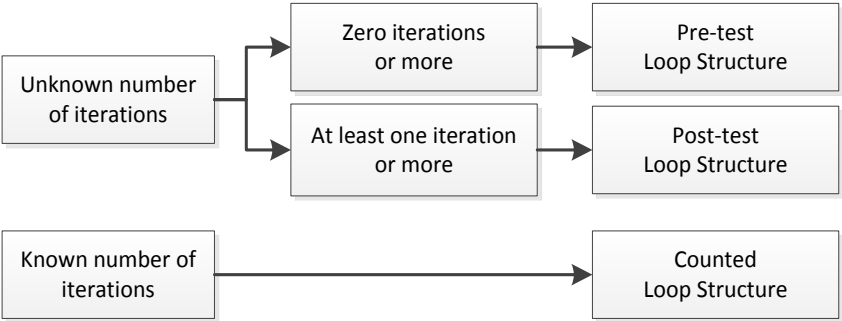
This chapter is dedicated to teaching you some useful tips and tricks that can help you write “better” code. You should always keep them in mind when you design your own algorithms, or even your own PHP scripts.

These tips and tricks can help you increase your code’s readability, help you choose which loop control structure is better to use in each given problem, and help make the code shorter or even faster. Of course there is no single perfect methodology because on one occasion the use of a specific tip or trick may help, but on another occasion the same tip or trick may have exactly the opposite result. Most of the time, code optimization is a matter of programming experience.

Remember! *Smaller algorithms are not always the best solution to a given problem. In order to solve a specific problem, you might write a very short algorithm that unfortunately proves to consume a lot of CPU time. On the other hand, you may solve the same problem with another algorithm which, even though it seems longer, calculates the result much faster.*

29.2 Choosing a Loop Control Structure

The following diagram can help you choose which loop control structure is better to use in each given problem, depending on the number of iterations.



Notice: *This diagram recommends the best option, not the only option. For example, when the number of iterations is known, it is not wrong to use a pre-test or a post-test loop structure instead. The proposed counted loop structure, though, is better since it is more convenient.*

29.3 The “Ultimate” Rule

One question that often preys on programmers’ minds when using pre-test or post-test loop structures, is how to determine which statements should be written inside, and which outside, the loop control structure and in which order.

There is one simple yet powerful rule—the “Ultimate” rule! Once you follow it, the potential for making a logic error is reduced to zero!

The “Ultimate” rule states:

- The variable or variables that participate in a loop's Boolean expression should be initialized before entering the loop.
- The value of the variable or variables that participate in a loop's Boolean expression should be updated (altered) within the loop and just before the loop's Boolean expression is evaluated.

For example, if variable `$x` is the variable that participates in a loop's Boolean expression, a pre-test loop structure should always be in the following form,

```
initialize $x
while ( Boolean_Expression ($x) ) {
    A statement or block of statements
    Update/alter $x
}
```

and a post-test loop structure should always be in the following form,

```
initialize $x
do {
    A statement or block of statements
    Update/alter $x
} while ( Boolean_Expression ($x) )
```

where

- `initialize $x` is any statement that assigns an initial value to variable `$x`. It can be either an input statement such as `trim(fgets(STDIN))`, or an assignment statement using the assignment operator (`=`). In a post-test loop structure though, this statement may sometimes be redundant and can be omitted since initialization of `$x` can occur directly inside the loop.
- `Boolean_Expression ($x)` can be any Boolean expression from a simple to a complex one, dependent on variable `$x`.
- `Update/alter $x` is any statement that alters the value of `$x` such as an input statement `trim(fgets(STDIN))`, an assignment statement using the assignment operator (`=`), or even incrementing/decrementing operators or compound assignment operators. It should always be the last statement in the loop.

Following are some examples that use the “Ultimate” rule.

Example 1

```
$a = trim(fgets(STDIN));           //Initialization of $a
while ($a > 0) {
    echo $a;
    $a = $a - 1;                   //Update/alteration of $a
}
```

Example 2

```
$s = 0;
```

```
$y = trim(fgets(STDIN));           //Initialization of $y
while ($y != -99) {
    $s = $s + $y;
    $y = trim(fgets(STDIN));       //Update/alteration of $y
}
```

Example 3

```
$s = 0;                             //Initialization of $s
do {
    $y = trim(fgets(STDIN));
    $s = $s + $y;                   //Update/alteration of $s
} while ($s < 1000);
```

Example 4

```
$y = 0;                             //Initialization of $y
do {
    $y = trim(fgets(STDIN));       //Update/alteration of $y
} while ($y < 0);
```

In this example, though, initialization of variable `$y` outside the loop is redundant and can be omitted since initialization can also be done inside the loop, as follows.

```
do {
    $y = trim(fgets(STDIN));       //Initialization and update/alteration of
    $y
} while ($y < 0);
```

Example 5

```
<?php
    $odd = 0;                       //Initialization of $odd
    $seven = 0;                    //Initialization of $seven
    while ($odd + $seven < 5) {
        $x = trim(fgets(STDIN));
        if ($x % 2 == 0) {
            $odd++;                //Update/alteration of $odd
        }
        else {
            $seven++;              //Update/alteration of $seven
        }
    }
    echo "Odds: ", $odd, "\nEvens: ", $seven;
?>
```

Now, you will realize why you should always follow the “Ultimate” rule!” Let’s say that the wording of this fifth example was:

Write a PHP script that lets the user enter five numbers and then displays the total number of odds and evens given.

This exercise was given to a class, and a student gave the following PHP script as an answer.

```
<?php
    $even = 0;
    $odd = 0;
    $x = trim(fgets(STDIN));
    while ($even + $odd < 5) {
        if ($x % 2 == 0) {
            $even++;
        }
        else {
            $odd++;
        }
        $x = trim(fgets(STDIN));
    }
    echo "Odds: ", $odd, "\nEvens: ", $even;
?>
```

At first sight the script looks correct, but it is also true that it contains a logic error, and unfortunately a difficult one. Can you spot it?

If you try to follow the flow of execution, you can confirm by yourself that the script runs smoothly—so smoothly that it makes you wonder if this book is reliable or if you should throw it away!

The problem becomes clear only when you try to enter all five of the expected values. The trace table that follows can help you determine where the problem lies. Assume that the user wants to enter the values 5, 10, 2, 4, and 20.

Step	Statement	Notes	\$even	\$odd	\$x
1	\$even = 0		0	?	?
2	\$odd = 0		0	0	?
3	\$x = trim(fgets(STDIN))	User enters the value 5	0	0	5
4	while (\$even + \$odd < 5)	This evaluates to true			
5	if (\$x % 2 == 0)	This evaluates to false			
6	\$odd++		0	1	5
7	\$x = trim(fgets(STDIN))	User enters the value 10	0	1	10
8	while (\$even + \$odd < 5)	This evaluates to true			

9	if (\$x % 2 == 0)	This evaluates to true			
10	\$seven++		1	1	10
11	\$x = trim(fgets(STDIN))	User enters the value 2	1	1	2
12	while (\$odd + \$seven < 5)	This evaluates to true			
13	if (\$x % 2 == 0)	This evaluates to true			
14	\$seven++		2	1	2
15	\$x = trim(fgets(STDIN))	User enters the value 4	2	1	4
16	while (\$odd + \$seven < 5)	This evaluates to true			
17	if (\$x % 2 == 0)	This evaluates to true			
18	\$seven++		3	1	4
19	\$x = trim(fgets(STDIN))	User enters the value 6	3	1	20
20	while (\$odd + \$seven < 5)	This evaluates to true			
21	if (\$x % 2 == 0)	This evaluates to true			
22	\$seven++		4	1	20
23	\$x = trim(fgets(STDIN))	User enters the value ??????????	4	1	????

And here is the logic error! At step 23, the computer asks the user to enter an additional sixth number!

Remember! You needed a PHP script that lets the user enter five numbers, not six!

One could say, “Okay, let’s change the loop’s Boolean expression from `$seven + $odd < 5` to `$seven + $odd < 4`. This can help!” Unfortunately, this change makes things worse, because now the flow of execution exits the loop earlier, before the last given value, 20, can be counted as even.

This is why you should always go by the book! Try to write Example 5 (not the student’s version) into NetBeans IDE and see how perfectly it operates, prompting the user to enter five and not six numbers!

29.4 Breaking Out of a Loop

Loops can consume too much CPU time so you have to be very careful when you use them. There are times when you need to break out of, or end, a loop before it completes all of its iterations, usually when a specified condition is met.

Suppose there is a counted loop structure that searches for a given letter within a string, as shown in the next PHP script.

```
<?php
    $s = "I have a dream";
```

```
echo "Enter a letter to search: ";
$letter = trim(fgets(STDIN));

$found = false;
for ($i = 0; $i <= strlen($s) - 1; $i++) {
    if ($s[$i] == $letter) {
        $found = true;
    }
}

if ($found == true) {
    echo "Letter ", $letter, " found!";
}

?>
```

Now suppose that the user enters the letter “h”. As you already know, the counted loop structure iterates a specified number of times, and it doesn’t care if the letter is actually found or not. Even though the letter “h” does exist at the second position of variable `$s`, the loop unfortunately continues to iterate until the end of the string, thus wasting CPU time.

Someone may say “*So what? Variable `$s` contains just 14 characters. No big deal!*” But it is a big deal, actually! In large scale data processing everything counts, so you should be very careful when using loop control structure, especially those that iterate too many times.

There are two approaches that can help you make scripts like the previous one run faster. The main idea, in both of them, is to break out of the loop when a specified condition is met; in this case when the given letter is found.

First Approach – Using the `break` statement

You can break out of a loop before it actually completes all of its iterations by using the `break` statement. If this statement sounds familiar to you, you are right, you have already met it; it usually goes along with the `switch` statement.

Look at the following PHP script. When the given letter is found within variable `$s`, the flow of execution immediately exits (breaks out of) the loop.

file_29.4a.php

```
<?php
$s = "I have a dream";

echo "Enter a letter to search: ";
$letter = trim(fgets(STDIN));

$found = false;
```



```
for ($i = 0; $i <= strlen($s) - 1; $i++) {  
    if ($s[$i] == $letter) {  
        $found = true;  
        break;  
    }  
}  
  
if ($found == true) {  
    echo "Letter ", $letter, " found!";  
}  
?>
```

Second Approach – Using a flag

This book tries to use the `break` statement as little as possible, and only in certain situations in which it really helps you make your code better. That's because the `break` statement doesn't actually exist in all computer languages; and since this book's intent is to teach you "Algorithmic Thinking" (and not just special statements that only PHP supports), let's look at an alternate approach.

To break out of a loop using a flag, there is one methodology that you can always follow: the counted loop structure must be converted to a pre-test loop structure, and the Boolean expression of the loop must be modified to evaluate an extra variable. In the following PHP script this extra variable is the variable `$found` which, when the given letter is found within variable `$s`, forces the flow of execution to immediately exit the loop.

file_29.4b.php

```
<?php  
$s = "I have a dream";  
  
echo "Enter a letter to search: ";  
$letter = trim(fgets(STDIN));  
  
$found = false;  
$i = 0;  
while ($i <= strlen($s) - 1 && $found == false) {  
    if ($s[$i] == $letter) {  
        $found = true;  
    }  
    $i++;  
}  
if ($found == true) {  
    echo "Letter ", $letter, " found!";  
}
```

```
}  
?>
```

Notice: *Imagine variable `$found` as a flag. Initially, the flag is not “raised” (`$found = false`). The flow of execution enters the loop, and the loop continues iterating as long as the flag is down (while ... `$found == false`). When something (usually a condition) raises the flag (assigns value `true` to variable `$found`), the flow of execution exits the loop.*

29.5 Cleaning Out Your Loops

As you already know, loops can consume too much CPU time, so you must be very careful and use them sparingly. Although a large number of iterations is sometimes inevitable, there are always things that you can do to make your loops perform better.

The next PHP script calculates the average value of the numbers 1, 2, 3, 4, 5, ... 10000.

```
<?php  
$s = 0;  
$i = 1;  
do {  
    $total = 10000;  
    $s = $s + $i;  
    $i++;  
} while ($i <= $total);  
  
$average = $s / $total;  
echo $average;  
?>
```

What you should always keep in mind when using loops, especially those that perform many iterations, is to avoid putting any statement inside a loop that serves no purposes in that loop. In the previous example, the statement `$total = 10000` is such a statement. Unfortunately, as long as it exists inside the loop, the computer executes it 10000 times for no reason, which of course affects the computer’s performance.

To resolve this problem, you can simply move this statement outside the loop, as follows.

```
<?php  
$s = 0;  
$i = 1;  
$total = 10000;  
do {  
    $s = $s + $i;
```

```
    $i++;  
} while ($i <= $total);  
  
$average = $s / $total;  
echo $average;  
?>
```

Exercise 29.5-1 *Cleaning Out the Loop*

The following code fragment calculates the average value of numbers 1, 2, 3, 4, ... 10000. Try to move as many statements as possible outside the loop to make the script more efficient.

```
$s = 0;  
for ($i = 1; $i <= 10000; $i++) {  
    $s = $s + $i;  
    $average = $s / 10000;  
}  
echo $average;
```

Solution

One very common mistake that novice programmers make when calculating average values is to put the statement that calculates the average inside the loop. Think about it! Imagine that you want to calculate your average grade in school. Your first step would be to calculate the sum of the grades for all of the courses that you're taking. Then, when all your grades have been summed up, you would divide that sum by the number of courses that you're taking.

Notice: *Calculating an average is a two-step process.*

Therefore, it is pointless to calculate the average value inside the loop. You can move this statement outside and right after the loop, and leave the loop just to sum up the numbers as follows.

```
$s = 0;  
for ($i = 1; $i <= 10000; $i++) {  
    $s = $s + $i;  
}  
$average = $s / 10000;  
echo $average;
```

Exercise 29.5-2 *Cleaning Out the Loop**The next formula*

$$S = \frac{1}{1^1 + 2^2 + 3^3 + \dots + N^N} + \frac{2}{1^1 + 2^2 + 3^3 + \dots + N^N} + \dots + \frac{N}{1^1 + 2^2 + 3^3 + \dots + N^N}$$

is solved using the following PHP script, where N is given by the user.

```
<?php
echo "Enter N: ";
$n = trim(fgets(STDIN));
$s = 0;
for ($i = 1; $i <= $n; $i++) {
    $denom = 0;
    for ($j = 1; $j <= $n; $j++) {
        $denom += pow($j, $j);
    }
    $s += $i / $denom;
}
echo $s;
?>
```

This code fragment
calculates the denominator.

Try to move as many statements as possible outside the loop to make the script more efficient.

Solution

As you can see from the formula, the denominator is common for all fractions. Thus, it is pointless to calculate it again and again for every fraction. You can calculate the denominator just once and use the result many times, as follows.

```
<?php
echo "Enter N: ";
$n = trim(fgets(STDIN));

$denom = 0;
for ($j = 1; $j <= $n; $j++) {
    $denom += pow($j, $j);
}

$s = 0;
for ($i = 1; $i <= $n; $i++) {
    $s += $i / $denom;
}
echo $s;
```

```
?>
```

29.6 Endless Loops and How to Avoid Them

All loop control structures must include a way to stop endless iterations from occurring within them. This means that there should be something inside the loop that eventually makes the flow of execution exit the loop.

The next example contains an endless loop (also known as an infinite loop). Unfortunately, the programmer forgot to increase variable `$i` inside the loop; therefore, variable `$i` can never reach the value 10.

```
$i = 1;
while ($i != 10) {
    echo "Hello there!";
}
```

Remember! *If a loop cannot stop iterating, it is called an endless loop or an infinite loop.*

An endless loop continues to iterate forever and the only way to stop it from iterating is to use magic forces! For example, when an application in a Windows operating system “hangs” (probably because the flow of execution entered an endless loop), the user must use the key combination ALT+CTRL+DEL, which forces the application to end. In NetBeans IDE, on the other hand, when you accidentally write and execute an endless loop, you can just click the “Stop”  button in the output window and the PHP interpreter immediately stops any action.

So, always remember to include at least one such statement that makes the flow of execution exit the loop. But still, this is not always enough! Take a look at the following code fragment.

```
$i = 1;
while ($i != 10) {
    echo "Hello there!";
    $i += 2;
}
```

Even though this code fragment does contain one such statement (`$i += 2`), unfortunately the flow of execution never exits the loop because the value 10 is never assigned to variable `$i`.

One thing that you can do to avoid this type of mistake is to never check the *counter* variable using `==` and `!=` comparison operators, especially in cases in which the *counter* variable increments or decrements by a value other than 1. You can use `<`, `<=`, `>`, and `>=` comparison operators instead; they guarantee that the flow of execution exits the loop when the variable *counter* exceeds termination value. The previous example can be fixed by replacing the `!=` with a `<`, or even a `<=` comparison operator.

```
$i = 1;
while ($i < 10) {
```

```
echo "Hello there!";  
$i += 2;  
}
```

29.7 Converting from a Counted Loop Structure to a Pre-Test Loop Structure

The conversion from a counted loop structure to a pre-test loop structure is quite easy. All you need to do is get the *counter* variable of the *for* statement and use it within the *while* statement, as follows.

Counted Loop Structure

```
for (counter = initial_value; counter OP1 final_value; counter = counter OP2 offset) {  
    A statement or block of statements  
}
```



Pre-Test Loop Structure

```
counter = initial_value;  
while (counter OP1 final_value) {  
    A statement or block of statements  
    counter = counter OP2 offset;  
}
```

where

- *counter* must always be a variable.
- *initial_value*, *final_value*, and *offset* can be a constant value or even a variable or an expression.
- *OP1* must be `<=` when *counter* increments and `>=` when *counter* decrements
- *OP2* must be `+` when *counter* increments and `-` when *counter* decrements

Remember! Be careful not to forget to include the statement that increments or decrements the *counter* variable at the end of the pre-test loop structure, because otherwise the loop will iterate endlessly!

Exercise 29.7-1 Converting the PHP Script

Rewrite the following PHP script using the pre-test loop structure.

```
<?php  
$x = 2;  
  
for ($i = -4; $i <= 4; $i = $i + 1) {
```

```
$x = pow($x, 2);
}

echo $x;
?>
```

Solution

Following is the script of the exercise, written again, with some notes on it.

file_29.7-1a.php

```
<?php
```

```
$x = 2;
```

```
counter: $i
```

```
OP1: <=
```

```
offset: 1  
OP2: +
```

```
for ($i = -4;
```

```
$i <= 4;
```

```
$i = $i + 1) {
```

```
initial_value: -4
```

```
final_value: 4
```

```
$x = pow($x, 2);
}
```

```
echo $x;
```

```
?>
```

Now, with everything clarified, the PHP script can be rewritten using the pre-test loop structure.

file_29.7-1b.php

```
<?php
```

```
$x = 2;
```

```
$i = -4;
```

```
while ($i <= 4) {
```

```
$x = pow($x, 2);
```

```
$i = $i + 1;
```

```
}
```

```
echo $x;
```

```
?>
```

If you type and execute both scripts in NetBeans IDE, you can find out for yourself that they both display the same result.

Exercise 29.7-2 Converting the PHP Script

Rewrite the following PHP script using the pre-test loop structure.

```
<?php
$f = trim(fgets(STDIN));
$x = 3;

for ($j = 20; $j >= $f; $j -= 5) {
    $x = $x / 2;
}

echo $x;
?>
```

Solution

Here is the script of the exercise, written again, with some notes on it.

file_29.7-2a.php

```
<?php
$final = trim(fgets(STDIN));
$x = 3;

counter: $j
for ($j = 20;
    OP1: >=
    $j >= $f;
    offset: 5
    OP2: -
    $j -= 5) {

    $x = $x / 2;
}

echo $x;
?>
```

Now, with everything clarified, the PHP script can be rewritten using the pre-test loop structure.

file_29.7-2b.php

```
<?php
$f = trim(fgets(STDIN));
$x = 3;
```



```
$j = 20;
while ($j >= $f) {
    $x = $x / 2;
    $j -= 5;
}

echo $x;

?>
```

29.8 Converting from a Pre-Test Loop Structure to a Counted Loop Structure

In PHP, the conversion from a pre-test loop structure to a counted loop structure can be done very easily for any form of the `while` statement. However, you will need to be aware that in computer programming languages other than PHP, such as Visual Basic, Pascal, and Delphi (to name a few) you can do this type of conversion, but only under certain circumstances. Throughout the rest of this chapter, you will encounter situations in which conversion **should not always be carried out**.

Following is the general form of the pre-test loop structure, in which conversion can actually be carried out in the majority of the programming languages.

Pre-Test Loop Structure

```
counter = initial_value;
while (counter OP3 final_value1) {
    A statement or block of statements 1
    counter = counter OP2 offset;
    A statement or block of statements 2
}
```



Counted Loop Structure

```
for (counter = initial_value; counter OP1 final_value2; counter = counter OP2 offset)
{
    A statement or block of statements 1
    A statement or block of statements 2
}
```

where

- `counter` must always be a variable
- `initial_value`, `final_value1`, `final_value2`, and `offset` can be a constant value or even a variable or an expression. In the case of a variable or an

expression, however, the content or the result correspondingly must be invariable inside the loop.

- *OP1* must be `<=` when *counter* increments and `>=` when *counter* decrements
- *OP2* must be `+` when *counter* increments and `-` when *counter* decrements
- *OP3* can be any comparison operator (`<`, `>`, `<=`, `>=`, `==`, `!=`)
- any existing *counter* variable inside statement or block of statements 2 must be replaced by the expression *counter* *OP2* *offset*

Surely you are a little bit confused right now. Don't be! The next examples will help you put things in order.

Exercise 29.8-1 Converting the PHP Script

Rewrite the following PHP script using the counted loop structure.

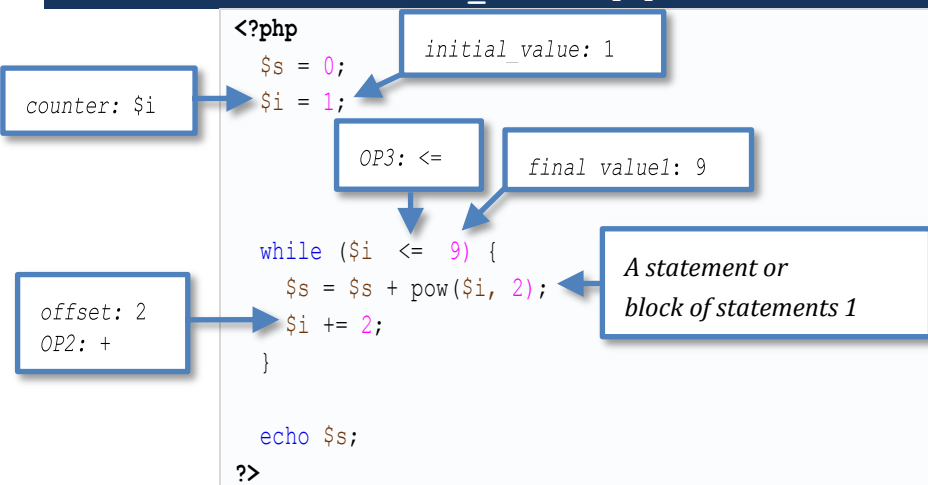
```
<?php
$s = 0;
$i = 1;
while ($i <= 9) {
    $s = $s + pow($i, 2);
    $i += 2;
}

echo $s;
?>
```

Solution

Following is the script of the exercise, written again, with some notes on it.

file_29.8-1a.php



Some extra notes:

- If you are wondering how to distinguish which variable is the *counter* of the pre-test loop structure, the answer is: *The counter is always the variable that the pre-test loop structure checks in every repetition.*
- Since *counter* increments, *OP1* must be `<=`.
- Since *offset* is 2 and variable `$i` increments, and since the loop's Boolean expression is `$i <= 9`, the last value of `$i` for which the pre-test loop structure performs an iteration is the value 9. Thus, *final_value2* must be 9.
- Inside statement or block of statements 1, variable *counter* must remain as is.

Now, with everything clarified, the PHP script can be rewritten using the counted loop structure.

file_29.8-1b.php

```
<?php
    $s = 0;
    for ($i = 1; $i <= 9; $i += 2) {
        $s = $s + pow($i, 2);
    }

    echo $s;

?>
```

If you type and execute both scripts in NetBeans IDE, you can find out for yourself that they both display the value 165.

Exercise 29.8-2 Converting the PHP Script

Rewrite the following PHP script using the counted loop structure.

```
<?php
    $s = 0;
    $y = 5;
    while ($y != -3) {
        $s = $s + 2 * $y;
        $y = $y - 2;
    }

    echo $s;

?>
```

Solution

Following is the script of the exercise, written again, with some notes on it.

file_29.8-2a.php

```
<?php
    $s = 0;
    $y = 5;
    while ($y != -3) {
        $s = $s + 2 * $y;
        $y = $y - 2;
    }

    echo $s;
?>
```

Annotations:

- initial_value: 5** points to `$y = 5;`
- counter: \$y** points to `$y = 5;`
- OP3: !=** points to the condition `$y != -3`
- final_value1: -3** points to the condition `$y != -3`
- A statement or block of statements 1** points to the loop body `$s = $s + 2 * $y; $y = $y - 2;`
- offset: 2** and **OP2: -** point to the expression `2 * $y` and the decrement `$y - 2` respectively.

Some extra notes:

- Since *counter* decrements, *OP1* must be `>=`.
- Since *offset* is 2 and variable *\$y* decrements, and since the loop's Boolean expression is `$y != -3`, it becomes clear that the last value of *\$y* for which the pre-test loop structure performs an iteration is the value -1. Thus, *final_value2* must be -1.
- Inside statement or block of statements 1, variable *counter* must remain as is.

Now, with everything clarified, the PHP script can be rewritten using the counted loop structure.

file_29.8-2b.php

```
<?php
    $s = 0;
    for ($y = 5; $y >= -1; $y = $y - 2) {
        $s = $s + 2 * $y;
    }
    echo $s;
?>
```

Exercise 29.8-3 Converting the PHP Script

Rewrite the following PHP script using the counted loop structure.

```
<?php
```

```
$s = 0;
```

```
$i = 1;
```

```
while ($i < 6) {
```

```
    $i++;
```

```
    $s = $s + pow($i, 2);
```

```
}
```

```
echo $s;
```

```
?>
```

Solution

Following is the script of the exercise, written again, with some notes on it.

file_29.8-3a.php

```
<?php
```

```
$s = 0;
```

```
$i = 1;
```

```
OP3: <
```

```
while ($i < 6) {
```

```
    $i++;
```

```
    $s = $s + pow($i, 2);
```

```
echo $s;
```

```
?>
```

initial_value: 1

counter: \$i

offset: 1

OP2: +

A statement or
block of statements 2

Some extra notes:

- The *counter* is variable `$i`.
- Since *counter* increments, *OP1* must be `<=`.
- Since *offset* is 1 and variable `$i` increments, and since the loop's Boolean expression is `$i < 6`, it becomes clear that the last value of `$i` for which the pre-test loop structure performs an iteration is the value 5. Thus *final_value2* must be 5.
- Inside statement or block of statements 2, variable *counter* must be replaced by the expression `counter OP2 offset`. Thus, the statement `$s = $s + pow($i, 2)` must become `$s = $s + pow($i + 1, 2)`.

Now, with everything clarified, the PHP script can be rewritten using the counted loop structure.

file_29.8-3b.php

```
<?php
    $s = 0;
    $i = 1;
    for ($i = 1; $i <= 5; $i++) {
        $s = $s + pow($i + 1, 2);
    }
    echo $s;
?>
```

If you type and execute both scripts in NetBeans IDE and execute them, you can find out for yourself that they both display the value 90.

Exercise 29.8-4 *Converting the PHP Script*

Rewrite the following PHP script using the counted loop structure.

```
<?php
    $y = 5;
    $x = 0;
    while ($y < 1000) {
        $x = $x + 2;
        $y = $y + $x;
    }
    echo $y;
?>
```

Solution

First of all, let's find out which of the two variables is the *counter* variable. The *counter* of the loop control structure is actually variable `$y`, and not variable `$x`, as you may mistakenly believe. It is easy to distinguish this because this is the variable that the pre-test loop structure checks in every repetition (`$y < 1000`).

Now, let's see what happens inside the loop. The *counter* variable `$y` increments by a value that is not constant because the value of variable `$x`, unfortunately, increments in every iteration! But as you have already been taught, the counted loop structure requires a constant offset and not a variable one! Therefore, that makes this PHP script unsuitable for conversion to a counted loop structure!

Notice: *If you think you understood everything about converting from a pre-test loop structure to a counted loop structure, you are very much mistaken! In PHP, previous conversion is actually possible, as shown here!*

```
?php
$x = 0;
for ($y = 5; $y < 1000; $y += $x) {
    $x = $x + 2;
}
echo $y;
?>
```

So, why does this book try to convince you that you should not make this type of conversion? The answer lies in the fact that this type of conversion cannot actually be done in every computer language. For example, you cannot do this conversion in languages such as Visual Basic, Pascal, Delphi, and many more.

If this example were written in Visual Basic, as shown next, it would be completely wrong!

```
x = 0
For y = 5 To 999 Step x
    x = x + 2
Next y
Console.WriteLine(y)
```

In Visual Basic, step (offset) is initially set to 0 and is never updated to any subsequent values of x. Thus, unfortunately, this code fragment performs infinite number of iterations.

Since this book aims to teach you the concepts of “Algorithmic Thinking” that apply to every computer language (not only to PHP), try to avoid using counted loop structures when a non-constant offset is required. Use a pre-test loop structure instead!

Exercise 29.8-5 Converting the PHP Script

Rewrite the following PHP script using the counted loop structure.

```
<?php
$s = 0;

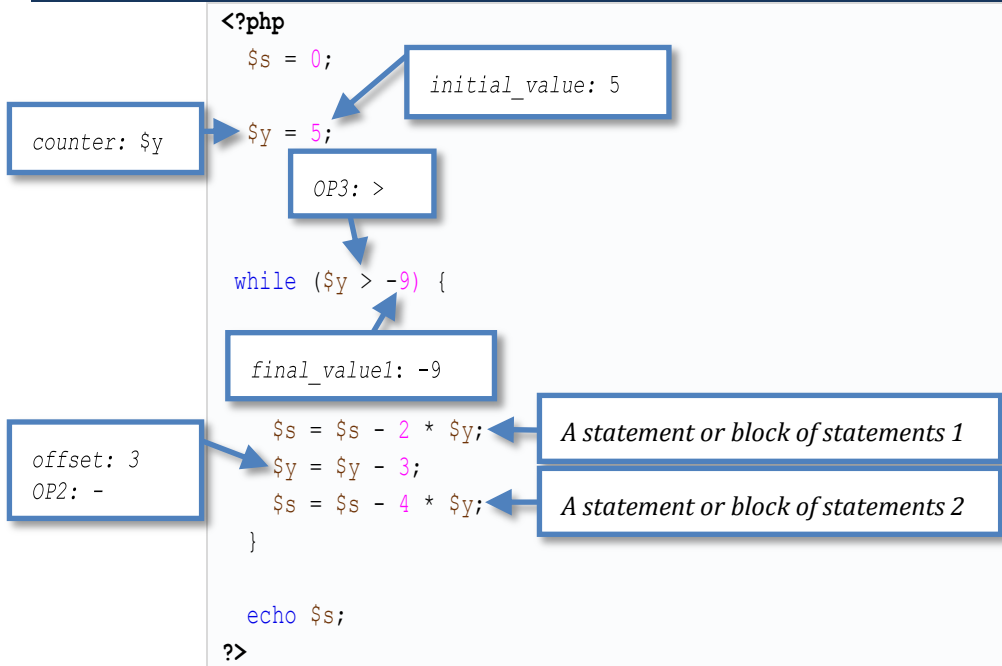
$y = 5;
while ($y > -9) {
    $s = $s - 2 * $y;
    $y = $y - 3;
}
```

```
$s = $s - 4 * $y;
}
echo $s;
?>
```

Solution

Following is the script of the exercise, written again, with some notes on it.

file_29.8-5a.php



Some extra notes:

- Since *counter* decrements, *OP1* must be `>=`.
- Since *offset* is 3 and variable *\$y* decrements and since the loop's Boolean expression is `$y > -9`, it becomes clear that the last value of *\$y* for which the pre-test loop structure performs an iteration is the value -7. Thus, *final_value2* must be -7.
- Inside statement or block of statements 1, variable *counter* must remain as is.
- Inside statement or block of statements 2, variable *counter* must be replaced by the expression *counter* *OP2* *offset*. Thus, the statement `$s = $s - 4 * $y` must become `$s = $s - 4 * ($y - 3)`.

Now, with everything clarified, the PHP script can be rewritten using the counted loop structure as follows.

file_29.8-5b.php

```
<?php
    $s = 0;

    for ($y = 5; $y >= -7; $y = $y - 3) {
        $s = $s - 2 * $y;
        $s = $s - 4 * ($y - 3);
    }

    echo $s;
?>
```

If you type both scripts into NetBeans IDE and execute them, they both display the value 90.

29.9 Converting from a Post-Test Loop Structure to a Pre-Test Loop Structure

Keep in mind that in a post-test loop structure the statement or block of statements of the structure are executed at least one time, as opposed to a pre-test loop structure, in which the statement or block of statements of the structure may never be executed.

In order to make a pre-test loop structure produce the same result(s) as a post-test loop structure, you need to write the statement or block of statements of the post-test loop structure twice, once before and once inside the pre-test structure, as follows.

Post-Test Loop Structure

```
do {
    A statement or block of statements
} while (Boolean_Expression)
```



Pre-Test Loop Structure

```
A statement or block of statements
while (Boolean_Expression) {
    A statement or block of statements
}
```

Exercise 29.9-1 Converting the PHP Script

Rewrite the following PHP script using the pre-test loop structure.

file_29.9-1a.php

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

do {
    $a += 2;
    echo $a, ":", $b;
    $b = $b + 5;
} while ($b <= $a);

?>
```

Solution

The conversion rule states that, in order to make a post-test loop structure produce the same result(s) as a pre-test loop structure, the statement or block of statements of the post-test loop structure must be written twice, once before and once inside the pre-test loop structure!

The next PHP script operates exactly the same way as the initial post-test loop!

file_29.9-1b.php

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

$a += 2;
echo $a, ":", $b;
$b = $b + 5;

while ($b <= $a) {
    $a += 2;
    echo $a, ":", $b;
    $b = $b + 5;
}

?>
```

You need to write this block of statements twice. Once before and once inside the pre-test loop structure!

Try to type and execute this script and the initial one in NetBeans IDE. You will find out for yourself that they both produce the same output results for any input values.

Now, let's try **not to use** the conversion rule you just learned, and see what happens. The next PHP script uses a pre-test loop structure and is supposed to produce the same results as the given post-test loop structure.

file_29.9-1c.php

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

while ($b <= $a) {
    $a += 2;
    echo $a, ":", $b;
    $b = $b + 5;
}
?>
```

Suppose the user enters the values 14 and 10 for variables `$a` and `$b` respectively. For these values, both scripts perform two iterations and both display 16:10 and 18:15 on screen. They seem to produce the same results, but do they?

Let's try another pair of input values. Suppose the user enters the values 2 and 3 for variables `$a` and `$b` respectively. In this case, the post-test loop structure performs one iteration as opposed to the pre-test loop structure, which never enters the loop. Thus, it performs zero iterations!

So, in order to avoid this kind of problem and to be sure that your conversions are always correct, you should always go by the book. In any case, this is what rules are for!

Exercise 29.9-2 *Converting the PHP Script*

Rewrite the following PHP script using the pre-test loop structure.

file_29.9-2a.php

```
<?php
$a = 0;

do {
    $b = trim(fgets(STDIN));
    $a = $a + $b;
} while ($a <= pow($b, 2));
echo $a;
?>
```

Solution

No questions, no buts. Just follow the rule! The PHP script using the pre-test loop structure becomes

file_29.9-2b.php

```
<?php
```

```
$a = 0;

$b = trim(fgets(STDIN));
$a = $a + $b;
while ($a <= pow($b, 2)) {
    $b = trim(fgets(STDIN));
    $a = $a + $b;
}
echo $a;

?>
```

Exercise 29.9-3 *Converting the PHP Script*

Rewrite the following PHP script using the pre-test loop structure.

file_29.9-3a.php

```
<?php
$a = 0;

do {
    echo $a;
    $b = trim(fgets(STDIN));
    $a = $a + $b;
} while ($a < 1000);

?>
```

Solution

Following the conversion rule, the PHP script using the pre-test loop structure becomes

file_29.9-3b.php

```
<?php
$a = 0;

echo $a;
$b = trim(fgets(STDIN));
$a = $a + $b;
while ($a < 1000) {
    echo $a;
    $b = trim(fgets(STDIN));
    $a = $a + $b;
}
```

```
?>
```

This script operates perfectly well, but an alternate and even better solution is presented next. Suppose you do not write the statement or block of statements twice but only once inside the pre-test structure, as follows.

file_29.9-3c.php

```
<?php
    $a = 0;

    while ($a < 1000) {
        echo $a;
        $b = trim(fgets(STDIN));
        $a = $a + $b;
    }
?>
```

As you can see, variable `$a` does not depend on user input (it is assigned the constant value 0). This means that the Boolean expression `($a < 1000)` definitely evaluates to `true` (when the script starts running) and therefore the flow of execution enters the loop. Thus, this pre-test loop performs at least one iteration (as the initial post-test loop actually does); therefore, you do not have to write the statement or block of statements twice.

In conclusion, in this particular exercise you can omit writing the block of statements outside the pre-test loop structure. This is because variable `$a` does not depend on user input and the Boolean expression `($a < 1000)` definitely evaluates to `true` (when the script starts running). They are, without doubt, redundant since the loop can do the whole job!

29.10 Converting from a Pre-Test Loop Structure to a Post-Test Loop Structure

You already know that a post-test loop first checks its Boolean expression and then may perform from zero to many iterations. In order to make a post-test loop structure produce the same result(s) as the pre-test loop structure, you must use an additional single-decision structure that checks the Boolean expression before entering the loop.

Pre-Test Loop Structure

```
while (Boolean_Expression) {
    A statement or block of statements
}
```



Post-Test Loop Structure

```
if (Boolean_Expression) {  
    do {  
        A statement or block of statements  
    } while (Boolean_Expression);  
}
```

Exercise 29.10-1 *Converting the PHP Script*

Rewrite the following PHP script using the post-test loop structure.

file_29.10-1a.php

```
<?php  
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
  
while ($b <= $a) {  
    $a *= 2;  
    $b = $b * 3;  
}  
echo $a, $b;  
?>
```

Solution

Following the conversion rule, the PHP script using the post-test loop structure becomes

file_29.10-1b.php

```
<?php  
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
  
if ($b <= $a) {  
    do {  
        $a *= 2;  
        $b = $b * 3;  
    } while ($b <= $a);  
}  
echo $a, $b;  
?>
```

Exercise 29.10-2 *Converting the PHP Script*

Rewrite the following PHP script using the post-test loop structure.

file_29.10-2a.php

```
<?php
    $p = 1;

    $a = trim(fgets(STDIN));
    while ($a != -1) {
        $p = $p * $a;
        $a = trim(fgets(STDIN));
    }
    echo $p;
?>
```

Solution

Following the conversion rule, the PHP script using the post-test loop structure becomes

file_29.10-2b.php

```
<?php
    $p = 1;

    $a = trim(fgets(STDIN));
    if ($a != -1) {
        do {
            $p = $p * $a;
            $a = trim(fgets(STDIN));
        } while ($a != -1);
    }
    echo $p;
?>
```

Exercise 29.10-3 *Converting the PHP Script*

Rewrite the following PHP script using the post-test loop structure.

file_29.10-3a.php

```
<?php
    $x = 0;
    $y = 1;
    while ($y < 10) {
```

```
$x = $x * 2;  
$y = $y + 1;  
}  
echo $x;  
?>
```

Solution

Following the conversion rule, the PHP script using the post-test loop structure becomes

file_29.10-3b.php

```
<?php  
$x = 0;  
$y = 1;  
if ($y < 10) {  
    do {  
        $x = $x * 2;  
        $y = $y + 1;  
    } while ($y < 10);  
}  
echo $x;  
?>
```

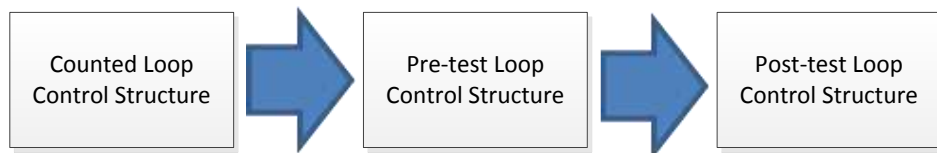
As you can see, variable `$y` does not depend on user input (it is assigned the constant value 1). This means that, whenever this script is executed, the Boolean expression (`$y < 10`) definitely evaluates to `true` and therefore the flow of execution enters the loop. Thus, in this case, you can actually omit the decision control structure. It is not wrong to leave it there, but it is without doubt redundant.

file_29.10-3c.php

```
<?php  
$y = 1;  
$x = 0;  
do {  
    $x = $x * 2;  
    $y = $y + 1;  
} while ($y < 10);  
echo $x;  
?>
```


29.11 Converting from a Counted Loop Structure to a Post-Test Loop Structure

This conversion doesn't have its own rules but you can use the rules you have already learned, that is, you can convert from a counted loop structure to an intermediate pre-test loop structure, and finally to a post-test loop structure, as shown in the next diagram.



For your convenience, everything you have learned so far is presented once again.

Counted Loop Structure

```

for (counter = initial_value; counter OP1 final_value; counter = counter OP2 offset) {
    A statement or block of statements
}
  
```



Pre-Test Loop Structure

```

counter = initial_value;
while (counter OP1 final_value) {
    A statement or block of statements
    counter = counter OP2 offset;
}
  
```



Post-Test Loop Structure

```

counter = initial_value;
if (counter OP1 final_value) {
    do {
        A statement or block of statements
        counter = counter OP2 offset;
    } while (counter OP1 final_value);
}
  
```

where

➤ *counter* must always be a variable.

- *initial_value*, *final_value* and *offset* can be a constant value or even a variable or an expression.
- *OP1* must be `<=` when *counter* increments and `>=` when *counter* decrements
- *OP2* must be `+` when *counter* increments and `-` when *counter* decrements

Exercise 29.11-1 Converting the PHP Script

Rewrite the following PHP script using the post-test loop structure.

file_29.11-1a.php

```
<?php
$start = trim(fgets(STDIN));
$end = trim(fgets(STDIN));
$x = 5;

for ($i = $start; $i <= $end; $i++) {
    $x = pow($x, 1.5);
}
echo $x;
?>
```

Solution

The intermediate pre-test loop structure is as follows.

file_29.11-1b.php

```
<?php
$start = trim(fgets(STDIN));
$end = trim(fgets(STDIN));
$x = 5;

$i = $start;
while ($i <= $end) {
    $x = pow($x, 1.5);
    $i++;
}
echo $x;
?>
```

and the final post-test loop structure is

file_29.11-1c.php

```
<?php
$start = trim(fgets(STDIN));
$end = trim(fgets(STDIN));
```

```
$x = 5;

$i = $start;
if ($i <= $end) {
    do {
        $x = pow($x, 1.5);
        $i++;
    } while ($i <= $end);
}
echo $x;

?>
```

Exercise 29.11-2 *Converting the PHP Script*

Rewrite the following PHP script using the post-test loop structure.

file_29.11-2a.php

```
<?php
$x = 10;

for ($i = -4; $i <= 4; $i++) {
    $x = pow($x, 2);
}

echo $x;

?>
```

Solution

The intermediate pre-test loop structure is as follows.

file_29.11-2b.php

```
<?php
$x = 10;

$i = -4;
while ($i <= 4) {
    $x = pow($x, 2);
    $i++;
}

echo $x;

?>
```

and the post-test loop structure is

file_29.11-2c.php

```
<?php
    $x = 10;

    $i = -4;
    if ($i <= 4) {
        do {
            $x = pow($x, 2);
            $i++;
        } while ($i <= 4);
    }

    echo $x;
?>
```

In this case, since variable `$i` does not depend on user input (it is assigned the constant value -4), the decision structure is redundant and can be omitted.

file_29.11-2d.php

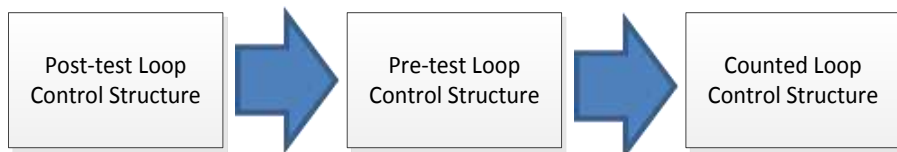
```
<?php
    $x = 10;

    $i = -4;
    do {
        $x = pow($x, 2);
        $i++;
    } while ($i <= 4);

    echo $x;
?>
```

29.12 Converting from a Post-Test Loop Structure to a Counted Loop Structure

This conversion doesn't have its own rules but you can use the rules you have already learned, that is, you can convert from a post-test loop structure to an intermediate pre-test loop structure, and finally to a counted loop structure, as shown in next diagram.



Keep in mind that in computer programming languages other than PHP, such as Visual Basic, Pascal, and Delphi (to name a few), you can do this type of conversion, but only under certain circumstances. The general form of the post-test loop structure, in which conversion can actually be carried out in the majority of the programming languages, is given next.

Post-Test Loop Structure

```
counter = initial_value;  
do {  
    A statement or block of statements 1  
    counter = counter OP2 offset;  
    A statement or block of statements 2  
} while (counter OP3 final_value1);
```



Pre-Test Loop Structure

```
counter = initial_value;  
A statement or block of statements 1  
counter = counter OP2 offset;  
A statement or block of statements 2  
while (counter OP3 final_value1) {  
    A statement or block of statements 1  
    counter = counter OP2 offset;  
    A statement or block of statements 2  
}
```



Counted Loop Structure

```
counter = initial_value;  
A statement or block of statements 1  
counter = counter OP2 offset;  
A statement or block of statements 2  
for (counter = initial_value OP2 offset;  
     counter OP1 final_value2;  
     counter = counter OP2 offset) {  
  
    A statement or block of statements 1  
    A statement or block of statements 2  
}
```

where

- *counter* must always be a variable.
- *initial_value*, *final_value1*, *final_value2* and *offset* can be a constant value or even a variable or an expression. In case of a variable or an expression, however, the content or the result correspondingly must be invariable inside the loop.
- *OP1* must be `<=` when *counter* increments and `>=` when *counter* decrements.
- *OP2* must be `+` when *counter* increments and `-` when *counter* decrements.
- *OP3* can be any comparison operator (`<`, `>`, `<=`, `>=`, `==`, `!=`).
- inside statement or block of statements 2 (those being inside the loop), any existing *counter* variable must be replaced by the expression *counter* *OP2* *offset*

Exercise 29.12-1 Converting the PHP Script

Rewrite the following PHP script using the counted loop structure.

file_29.12-1a.php

```
<?php
$s = 0;
$a = trim(fgets(STDIN));
$i = $a;
do {
    $s = $s + pow($i, 2);
    $i = $i + 1;
} while ($i != 10);
echo $s;
?>
```

Solution

The intermediate pre-test loop structure is as follows.

file_29.12-1b.php

```
<?php
$s = 0;
$a = trim(fgets(STDIN));
$i = $a;
$s = $s + pow($i, 2);
$i = $i + 1;
while ($i != 10) {
    $s = $s + pow($i, 2);
    $i = $i + 1;
}
```

This is the block of statements of the post-test loop structure, here written twice; once before and once inside the pre-test loop structure

```
}  
echo $s;  
?>
```

and the final counted loop structure becomes

file_29.12-1c.php

```
<?php  
$s = 0;  
$a = trim(fgets(STDIN));  
$i = $a;  
$s = $s + pow($i, 2);  
$i = $i + 1;  
  
for ($i = $a + 1; $i <= 9; $i = $i + 1) {  
    $s = $s + pow($i, 2);  
}  
echo $s;  
?>
```

Exercise 29.12-2 *Converting the PHP Script*

Rewrite the following PHP script using the counted loop structure.

file_29.12-2a.php

```
<?php  
$s = 0;  
$a = trim(fgets(STDIN));  
$i = $a;  
do {  
    $s = $s + $i;  
    $i = $i + 3;  
    $s = $s + pow($i, 2);  
} while ($i < 60);  
echo $s;  
?>
```

Solution

The intermediate pre-test loop structure is as follows.

file_29.12-2b.php

```
<?php  
$s = 0;
```

```

$a = trim(fgets(STDIN));
$i = $a;
$s = $s + $i;
$i = $i + 3;
$s = $s + pow($i, 2);
while ($i < 60) {
    $s = $s + $i;
    $i = $i + 3;
    $s = $s + pow($i, 2);
}
echo $s;
?>

```

This is the block of statements of the post-test loop structure, here written twice; once before and once inside the pre-test loop structure

and the final counted loop structure becomes

file_29.12-2c.php

```
<?php
```

```

$s = 0;
$a = trim(fgets(STDIN));
$i = $a;
$s = $s + $i;
$i = $i + 3;
$s = $s + pow($i, 2);

```

initial_value OP2 offset

```

for ($i = $a + 3; $i <= 59; $i = $i + 3) {
    $s = $s + $i;

    $s = $s + pow($i + 3, 2);
}
echo $s;
?>

```

Actually, this is the statement or block of statements 2; thus, *counter* variable *\$i* has been replaced by the expression *counter OP2 offset*.

29.13 Using the “From Inner to Outer” Method in Loop Control Structures

The “from inner to outer” method has already been discussed in Chapter 21. This book tries to teach you “Algorithmic Thinking” by first designing the inner (nested) control structures. Then, as the algorithm is developed, more and more control structures are added, nesting all previous ones.

Let’s try the following example.

Write a PHP script that displays the following multiplication table as it is shown below.

1x1=1	1x2=2	1x3=3	1x4=4	1x5=5	1x6=6	1x7=7	1x8=8	1x9=9
2x1=2	2x2=4	2x3=6	2x4=8	2x5=10	2x6=12	2x7=14	2x8=16	2x9=18
3x1=3	3x2=6	3x3=9	3x4=12	3x5=15	3x6=18	3x7=21	3x8=24	3x9=27
4x1=4	4x2=8	4x3=12	4x4=16	4x5=20	4x6=24	4x7=28	4x8=32	4x9=36
5x1=5	5x2=10	5x3=15	5x4=20	5x5=25	5x6=30	5x7=35	5x8=40	5x9=45
6x1=6	6x2=12	6x3=18	6x4=24	6x5=30	6x6=36	6x7=42	6x8=48	6x9=54
7x1=7	7x2=14	7x3=21	7x4=28	7x5=35	7x6=42	7x7=49	7x8=56	7x9=63
8x1=8	8x2=16	8x3=24	8x4=32	8x5=40	8x6=48	8x7=56	8x8=64	8x9=72
9x1=9	9x2=18	9x3=27	9x4=36	9x5=45	9x6=54	9x7=63	9x8=72	9x9=81

According to the “from inner to outer” method, you start by writing the inner control structure and then, when everything is tested and operates fine, you can add the outer control structures.

So, let’s try to display only the first line of the multiplication table. Examination of this line reveals that, in each multiplication, the multiplicand is always 1. The loop control structure that displays only the first line of the multiplication table is as follows. Imagine a variable `$i` that contains the value 1.

```
for ($j = 1; $j <= 9; $j++) {  
    echo $i, "x", $j, "=", $i * $j, "\t";  
}
```

If you execute this code fragment, the result is

1x1=1	1x2=2	1x3=3	1x4=4	1x5=5	1x6=6	1x7=7	1x8=8	1x9=9

Remember! The special sequence of characters `\t` “displays” a tab character after each iteration. This ensures that everything is aligned properly.

The inner (nested) loop control structure is ready. What you need now is a way to execute this control structure nine times, but each time variable `$i` should contain a different value, from 1 to 9. This code fragment is as follows.

```
for ($i = 1; $i <= 9; $i++) {  
    //Here goes the code that displays one single line  
    //of the multiplication table  
    echo "\n";  
}
```

Notice: The echo `"\n"` statement is used to “display” a line break between lines.

Now, you can combine both code fragments, nesting the first into the second one. The final PHP script becomes

file_29.13.php

```
<?php  
for ($i = 1; $i <= 9; $i++) {
```

```
for ($j = 1; $j <= 9; $j++) {  
    echo $i, "x", $j, "=", $i * $j, "\t";  
}  
echo "\n";  
}  
??
```

29.14 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. When the number of iterations is unknown, you can use the counted loop structure.
2. When the number of iterations is known, you cannot use a post-test loop structure.
3. According to the “Ultimate” rule, in a pre-test loop structure, the initialization of the variable that participates in the loop’s Boolean expression should be done inside the loop.
4. According to the “Ultimate” rule, in a pre-test loop structure, the statement that updates/alters the value of the variable that participates in the loop’s Boolean expression must be the last statement within the loop.
5. According to the “Ultimate” rule, in a post-test loop structure, the initialization of the variable that participates in the loop’s Boolean expression can sometimes be done inside the loop.
6. According to the “Ultimate” rule, in a post-test loop structure, the update/alteration of the variable that participates in the loop’s Boolean expression must be the first statement within the loop.
7. You can break out of a loop before it completes all iterations using the `break` statement.
8. A statement that assigns a constant value to a variable is better placed inside a loop control structure.
9. In the following code fragment:

```
for ($i = 1; $i <= 30; $i++) {  
    $a = "Hello";  
    echo $a;  
}
```

there is at least one statement that can be moved outside the counted loop structure.

10. In the following code fragment:

```
$s = 0;  
$count = 1;  
while ($count < 100) {  
    $a = trim(fgets(STDIN));
```

```
$s += $a;  
$average = $s / $count;  
$count++;  
}  
echo $average;
```

there is at least one statement that can be moved outside the counted loop structure.

11. In the following code fragment:

```
$s = 0;  
$y = trim(fgets(STDIN));  
while ($y != -99) {  
    $s = $s + $y;  
    $y = trim(fgets(STDIN));  
}
```

there is at least one statement that can be moved outside the counted loop structure.

12. The following code fragment:

```
$i = 1;  
while ($i != 100) {  
    echo "Hello there!";  
    $i += 5;  
}
```

satisfies the property of finiteness.

13. When the not equal (`!=`) comparison operator is used in the Boolean expression of a pre-test loop structure, the loop always iterates endlessly.

14. The following code fragment:

```
$i = 0;  
do {  
    echo "Hello there!";  
    $i += 5;  
} while ($i < 100);
```

satisfies the property of finiteness.

15. The following code fragment:

```
for ($i = 0; $i <= 10; $i += 2) {  
    $x = sqrt($i, 2);  
    echo $x;  
}
```

cannot be converted to a pre-test loop structure.

16. The following code fragment:

```
$y = 0;  
$x = 0;  
while ($y < 1000) {  
    $y = $y + $x;  
    $x++;  
}
```

can be converted to a counted loop structure in all computer languages.

17. When converting from a post-test loop structure to a pre-test loop structure, there are cases in which you can omit writing the statement or block of statements of the post-test loop structure twice.
18. When converting from a pre-test loop structure to a post-test loop structure, there are cases in which you can omit the extra decision control structure.
19. The conversion from a counted loop structure to a post-test loop structure is possible but you need to convert to a pre-test loop structure first.

29.15 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. When the number of iterations is unknown, you can use
 - a. the pre-test loop structure.
 - b. the post-test loop structure.
 - c. all of the above
2. When the number of iterations is known, you can use
 - a. the pre-test loop structure.
 - b. the post-test loop structure.
 - c. the counted loop structure.
 - d. all of the above
3. According to the “Ultimate” rule, in a pre-test loop structure, the initialization of the variable that participates in the loop’s Boolean expression should be done
 - a. inside the loop.
 - b. outside the loop.
 - c. all of the above
4. According to the “Ultimate” rule, in a pre-test loop structure, the update/alteration of the variable that participates in the loop’s Boolean expression should be done
 - a. inside the loop.
 - b. outside the loop.
 - c. all of the above

5. According to the “Ultimate” rule, in a post-test loop structure, the update/alteration of the variable that participates in the loop’s Boolean expression should be done
- inside the loop.
 - outside the loop.
 - all of the above

6. In the following code fragment

```
$s = 0;
for ($i = 1; $i <= 100; $i++) {
    $s = $s + $i;
    $x = 100;
    $average = $s / 100;
}
```

the number of statements that can be moved outside of the counted loop structure is

- 0.
 - 1.
 - 2.
 - 3.
7. When this comparison operator is used in the Boolean expression of a post-test loop structure, the loop iterates forever.
- ==
 - <=
 - >=
 - it depends
8. The conversion from a counted loop structure to a pre-test loop structure
- cannot always be carried out.
 - can always be carried out.
 - can be carried out but the pre-test loop structure performs less iterations.
 - none of the above
9. In the majority of the programming languages, the conversion from a pre-test loop structure to a counted loop structure
- cannot always be carried out.
 - should not always be carried out.
 - can always be carried out.
 - can be carried out but the counted loop structure performs less iterations.
 - none of the above

10. The conversion from a post-test loop structure to a pre-test loop structure
 - a. cannot always be carried out.
 - b. can always be carried out.
 - c. can be carried out but the pre-test loop structure performs more iterations.
 - d. none of the above
11. The conversion from a pre-test loop structure to a post-test loop structure
 - a. cannot always be carried out.
 - b. can always be carried out.
 - c. can be carried out but the post-test loop structure performs more iterations.
 - d. none of the above
12. The conversion from a counted loop structure to a post-test loop structure
 - a. cannot always be carried out.
 - b. can always be carried out.
 - c. can be carried out but the post-test loop structure performs less iterations.
 - d. none of the above
13. In the majority of the programming languages, the conversion from a post-test loop structure to a counted loop structure
 - a. cannot always be carried out.
 - b. should not always be carried out.
 - c. can always be carried out.
 - d. can be carried out but the counted loop structure performs more iterations.
 - e. none of the above

29.16 Review Exercises

Complete the following exercises.

1. The following code fragment calculates the average value of 100 numbers entered by the user. Try to move as many statements as possible outside the loop to make it more efficient.

```
$s = 0;
for ($i = 1; $i <= 100; $i++) {
    $number = trim(fgets(STDIN));
    $s = $s + $number;
    $average = $s / 100;
}
echo $average;
```

2. The following formula

$$S = \frac{1}{1 \cdot 2 \cdot 3 \cdot \dots \cdot 100} + \frac{2}{1 \cdot 2 \cdot 3 \cdot \dots \cdot 100} + \dots + \frac{100}{1 \cdot 2 \cdot 3 \cdot \dots \cdot 100}$$

is solved using the following PHP script.

```
<?php
    $s = 0;
    for ($i = 1; $i <= 100; $i++) {
        $denom = 1;
        for ($j = 1; $j <= 100; $j++) {
            $denom *= $j;
        }
        $s += $i / $denom;
    }
    echo $s;
?>
```

Try to move as many statements as possible outside the loop to make it more efficient.

3. Rewrite the following code fragment using the pre-test loop structure.

```
$s = 10;
for ($i = 1; $i <= 10; $i++) {
    $s += sqrt($i);
}
echo $s;
```

4. Rewrite the following code fragment using the pre-test loop structure.

```
$start = trim(fgets(STDIN));
$end = trim(fgets(STDIN));
for ($i = $start; $i <= $end; $i++) {
    echo $i;
}
```

5. Rewrite the following code fragment using the counted loop structure.

```
$i = 100;
$s = 0;
while ($i > 0) {
    $s = $s + sqrt($i);
    $i -= 5;
}
echo $s;
```

6. Rewrite the following code fragment using the counted loop structure.

```
$s = 0;
$i = 1;
```

```
$y = 0;
while ($i <= 10) {
    $i++;
    $s = $s + sqrt($y + $i);
    $y = $y + $i * 2;
}
echo $s;
```

7. Rewrite the following code fragment using the counted loop structure.

```
$y = 0;
$i = 1;
while ($i < 10) {
    $a = trim(fgets(STDIN));
    $a += $i;
    $i += 2;
    $y = $y + pow($a + $i, 3);
}
echo $y;
```

8. Rewrite the following code fragment using the counted loop structure.

```
$y = 1;
$x = 0;
while ($y < 1000) {
    $x = pow($y, 2);
    $y = $y + $x;
}
echo $y;
```

9. Rewrite the following code fragment using the pre-test loop structure.

```
$s = 0;
$a = trim(fgets(STDIN));
do {
    $s += $a;
    $a = trim(fgets(STDIN));
} while ($a <= $s);
echo $s;
```

10. Rewrite the following code fragment using the pre-test loop structure.

```
$a = 100;
$count = 0;
do {
    echo $a;
    $b = trim(fgets(STDIN));
```



```
$count++;  
$a -= sqrt($b);  
} while ($a >= 0);  
echo $count;
```

11. Rewrite the following code fragment using the post-test loop structure.

```
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
while ($b <= 1000) {  
    $a += 2;  
    $b = $b * $a;  
    echo $b;  
}
```

12. Rewrite the following code fragment using the post-test loop structure.

```
$s = 0;  
$a = trim(fgets(STDIN));  
while ($a != -99) {  
    $s = $s + pow($a, 2);  
    $a = trim(fgets(STDIN));  
}  
echo $s;
```

13. Rewrite the following code fragment using the post-test loop structure.

```
$x = 0;  
$y = -10;  
while ($y < 10) {  
    $x = $x + pow(2, $y);  
    $y = $y + 1;  
}  
echo $x;
```

14. Rewrite the following code fragment using the post-test loop structure.

```
$start = trim(fgets(STDIN));  
$x = 1;  
for ($i = $start; $i <= $start * 2; $i++) {  
    $x = pow($x, 1.1) + $i;  
}  
echo $x;
```

15. Rewrite the following code fragment using the post-test loop structure.

```
$x = 42;  
for ($i = 1; $i <= 100; $i++) {
```

```
$x = sqrt($x) + $i;  
echo $x, "\n";  
}
```

16. Rewrite the following code fragment using the counted loop structure.

```
<?php  
$p = 1;  
$a = trim(fgets(STDIN));  
$i = $a;  
do {  
    $p = $p * pow($i, 2);  
    $i = $i + 5;  
    $p = $p + $i;  
} while ($i < 20);  
echo $p;  
?>
```

17. Rewrite the following code fragment using the counted loop structure.

```
<?php  
$start = trim(fgets(STDIN));  
$end = trim(fgets(STDIN));  
$x = 1000;  
$i = $start;  
if ($i <= $end) {  
    do {  
        $x = sqrt($x);  
        $i += 2;  
    } while ($i <= $end);  
}  
echo $x;  
?>
```

18. Write a PHP script that displays every combination of two integers as well as their resulting product, for pairs of integers between 1 and 4. The output should display as follows.

```
1 x 1 = 1  
1 x 2 = 2  
1 x 3 = 3  
1 x 4 = 4  
2 x 1 = 2  
2 x 2 = 4  
2 x 3 = 6
```

```
2 x 4 = 8
...
...
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
```

19. Write a PHP script that displays the multiplication table for pairs of integers between 1 and 12, as shown next. Please note that the output is aligned with tabs.

```
      |  1  2  3  4  5  6  7  8  9 10 11 12
-----
1 |  1  2  3  4  5  6  7  8  9 10 11 12
2 |  2  4  6  8 10 12 14 16 18 20 22 24
3 |  3  6  9 12 15 18 21 24 27 30 33 36
... | ...
11 | 11 22 33 44 55 66 77 88 99 110 121 132
12 | 12 24 36 48 60 72 84 96 108 120 132 144
```

Chapter 30

Flowcharts with Loop Control Structures

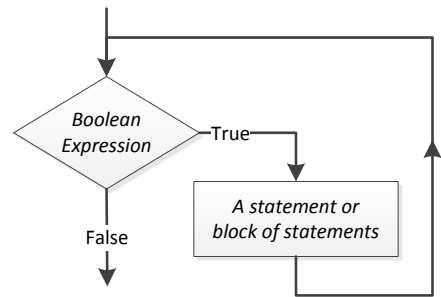
30.1 Introduction

By working through the previous chapters, you’ve become familiar with all the loop control structures, how to use them, and which to use in every case. You’ve learned about the “Ultimate” rule, how to break out of a loop, how to convert from one loop control structure to another, and much more. Since flowcharts are an ideal way to learn “Algorithmic Thinking” and to help you better understand specific control structures, this chapter will teach you how to convert a PHP script to a flowchart and vice versa, that is, a flowchart to a PHP script.

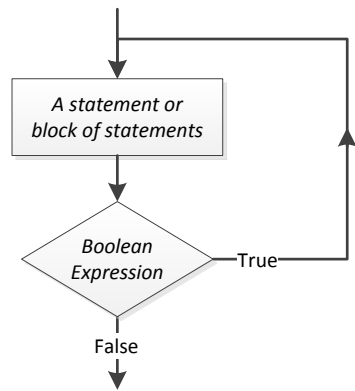
30.2 Converting PHP Scripts to Flowcharts

To convert a PHP script to a flowchart, you need to recall all loop control structures and their corresponding flowcharts. Following you will find them all summarized, plus a new one—the mid-test loop structure!

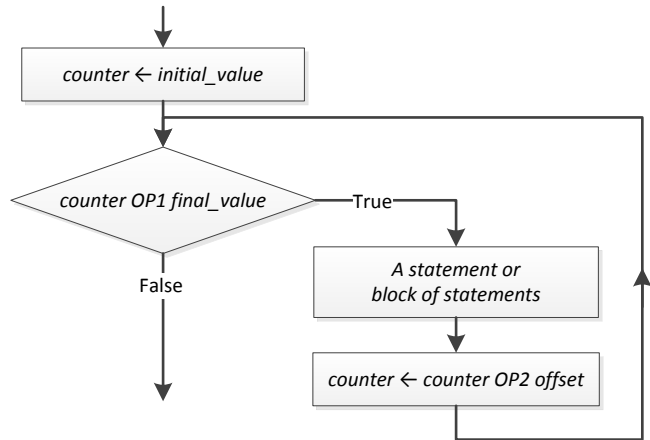
The Pre-Test Loop Structure



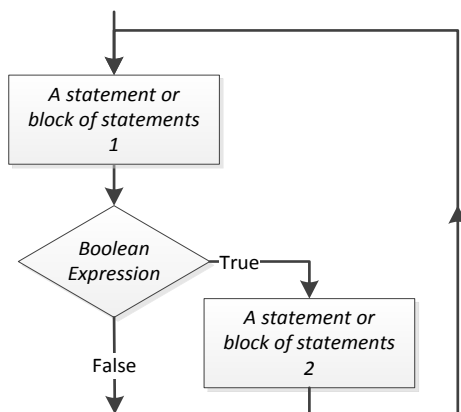
The Post-Test Loop Structure



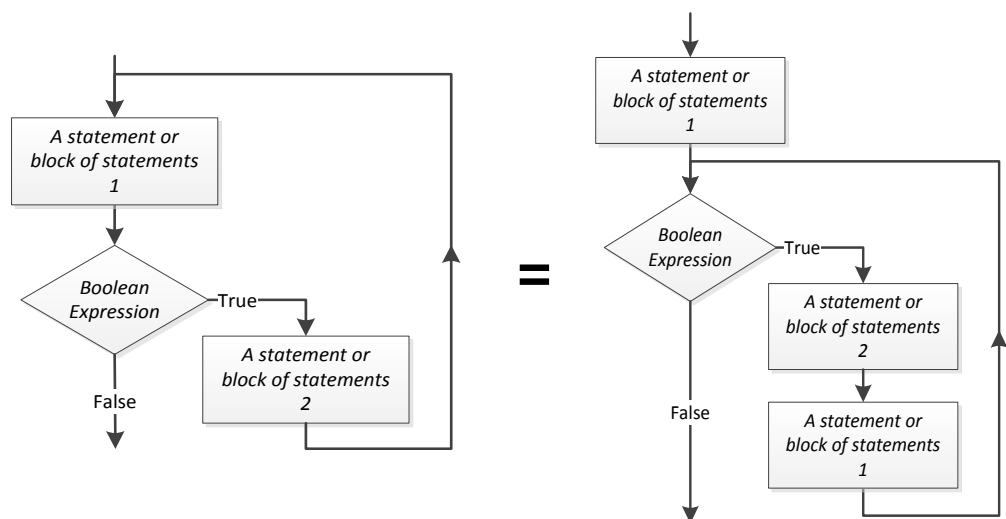
The Counted Loop Structure



The Mid-Test Loop Structure (this is a new one!)



Although this loop control structure is directly supported in some computer languages such as Ada, unfortunately this is not true for the majority of the programming languages. So, in order to write the corresponding PHP script, you have to convert this flowchart to something more familiar, as shown next. These two flowcharts are considered to be equivalent.



Next, you will find many exercises that can clarify things that you might still need help understanding.

Exercise 30.2-1 Designing the Flowchart

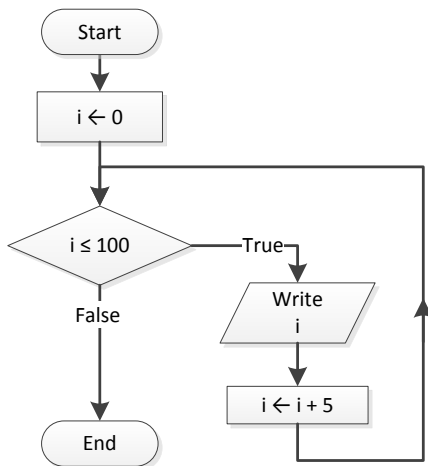
Design the flowchart that corresponds to the following PHP script.

```
<?php
```

```
$i = 0;
while ($i <= 100) {
    echo $i;
    $i += 5;
}
?>
```

Solution

This PHP script contains a pre-test loop structure. The corresponding flowchart that follows includes what you have been taught so far.



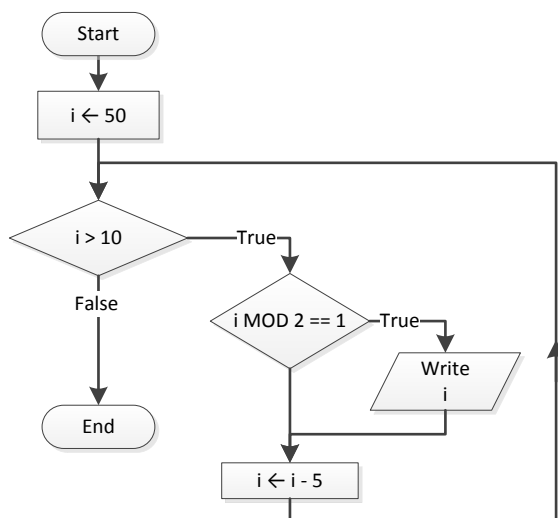
Exercise 30.2-2 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

```
<?php
    $i = 50;
    while ($i > 10) {
        if ($i % 2 == 1) {
            echo $i;
        }
        $i -= 5;
    }
?>
```

Solution

This PHP script contains a pre-test loop structure which nests a single-alternative decision structure. The corresponding flowchart that follows includes what you have been taught so far.



Exercise 30.2-3 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

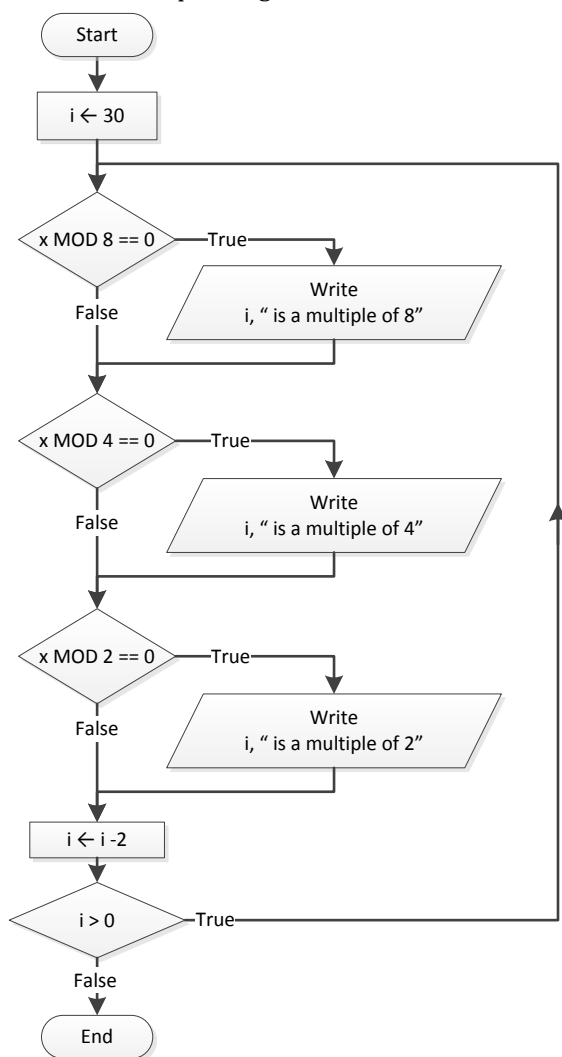
```

<?php
    $i = 30;
    do {
        if ($i % 8 == 0) {
            echo $i, " is a multiple of 8\n";
        }
        if ($i % 4 == 0) {
            echo $i, " is a multiple of 4\n";
        }
        if ($i % 2 == 0) {
            echo $i, " is a multiple of 2\n";
        }
        $i -= 2;
    } while ($i > 0);
?>

```

Solution

This PHP script contains a post-test loop structure that nests three single-alternative decision structures. The corresponding flowchart is as follows.



Exercise 30.2-4 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

```
<?php
for ($hour = 1; $hour <= 24; $hour++) {
    echo "Hour is ", $hour, ":00. ";
    if ($hour >= 4 && $hour < 12) {
        echo "Good Morning\n";
    }
}
```

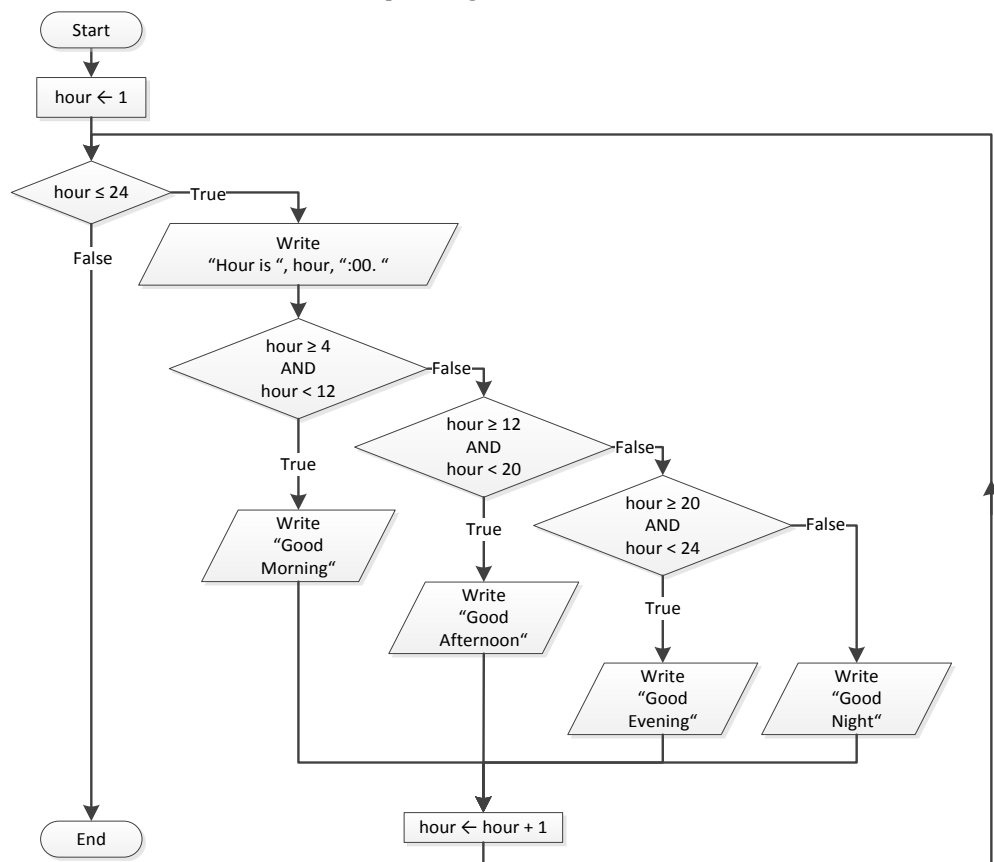
```

elseif ($hour >= 12 && $hour < 20) {
    echo "Good Afternoon\n";
}
elseif ($hour >= 20 && $hour < 24) {
    echo "Good Evening\n";
}
elseif ($hour <= 24) {
    echo "Good Night\n";
}
}
?>

```

Solution

This PHP script contains a counted loop structure that nests a multiple-alternative decision structure. The corresponding flowchart is as follows.



Exercise 30.2-5 *Designing the Flowchart*

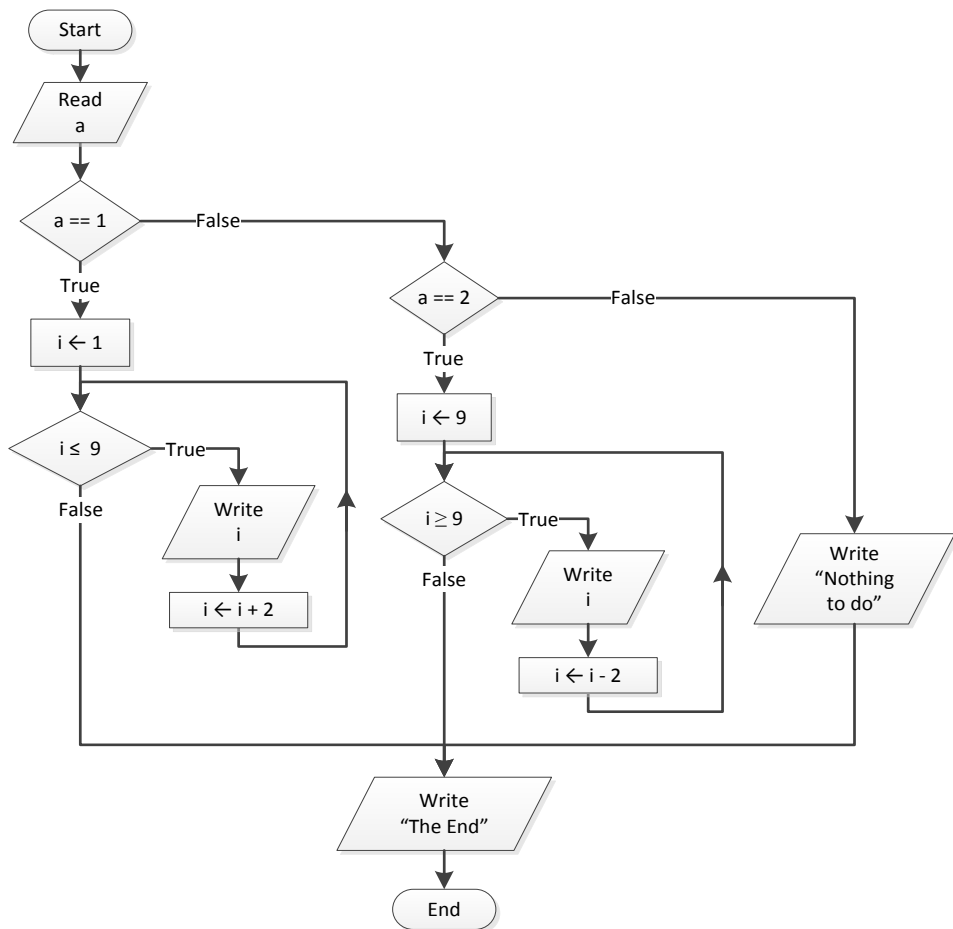
Design the flowchart that corresponds to the following PHP script.

```
<?php
$a = trim(fgets(STDIN));

switch ($a) {
    case 1:
        for ($i = 1; $i <= 9; $i += 2) {
            echo $i;
        }
        break;
    case 2:
        for ($i = 9; $i >= 1; $i -= 2) {
            echo $i;
        }
        break;
    default:
        echo "Nothing to do!";
}
echo "The End!";
?>
```

Solution

This PHP script contains a case decision structure that nests two counted loop structures. The corresponding flowchart is as follows.



Exercise 30.2-6 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

```

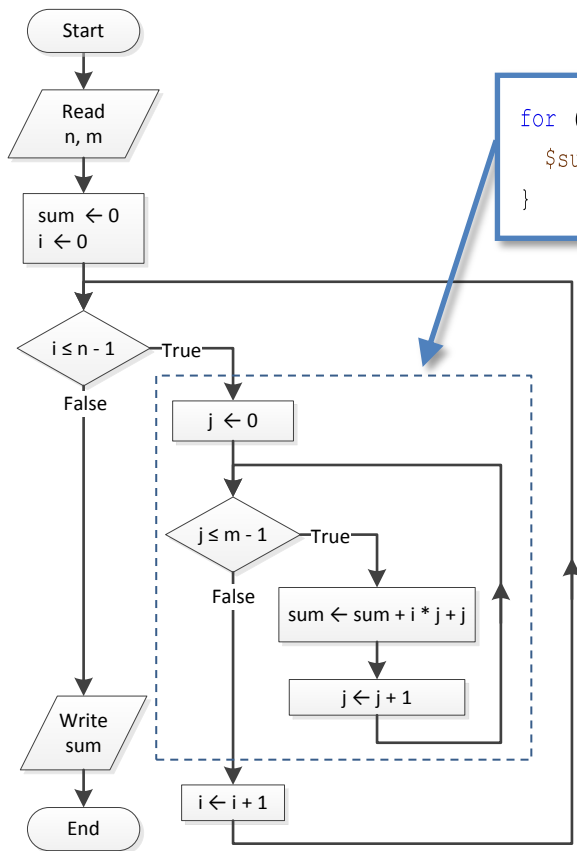
<?php
    $n = trim(fgets(STDIN));
    $m = trim(fgets(STDIN));

    $sum = 0;
    for ($i = 0; $i <= $n - 1; $i++) {
        for ($j = 0; $j <= $m - 1; $j++) {
            $sum += $i * $j + $j;
        }
    }
    echo $sum;
  
```

?>

Solution

This PHP script contains nested loop control structures; a counted loop structure nested within another counted loop structure. The corresponding flowchart is as follows.



```

for ($j = 0; $j <= $m - 1; $j++){
    $sum += $i * $j + $j;
}

```

Exercise 30.2-7 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

```
<?php
```

```

$s = 0;
for ($i = 0; $i <= 99; $i++) {
    $n = trim(fgets(STDIN));
    while ($n < 0) {
        echo "Error";
        $n = trim(fgets(STDIN));
    }
}

```

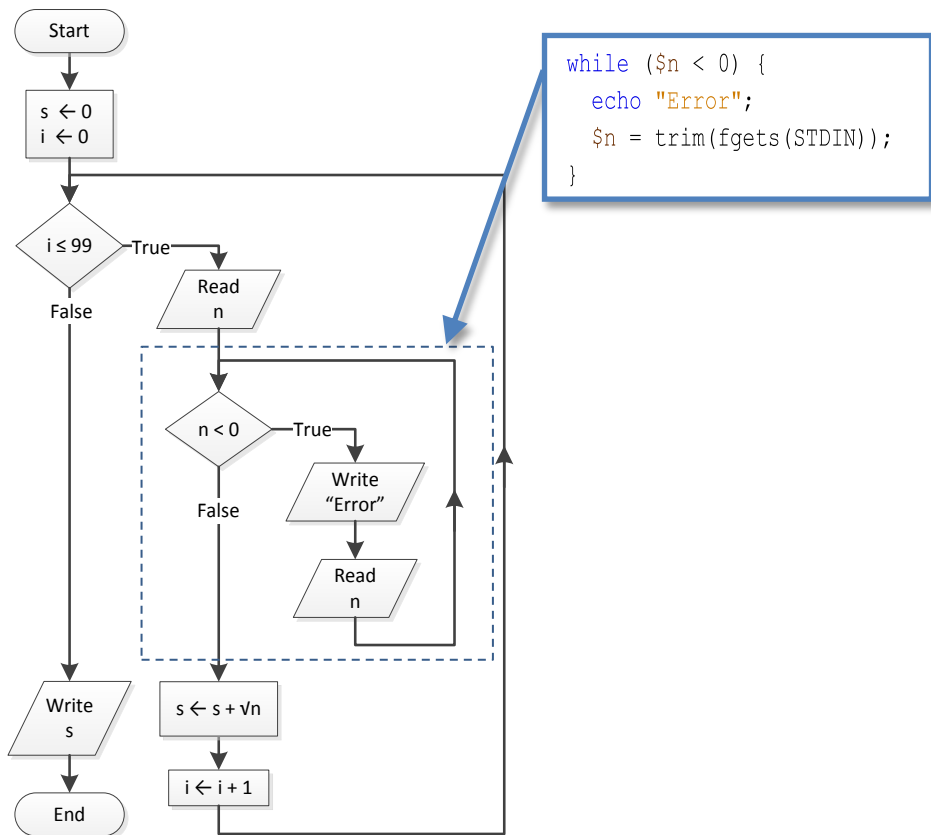
```

    }
    $s += sqrt($n);
  }
  echo $s;
?>

```

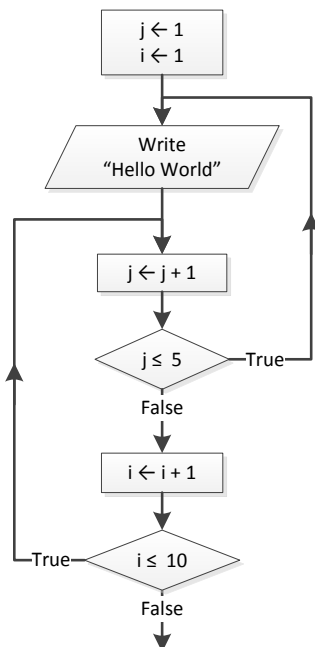
Solution

This PHP script contains nested loop control structures; a pre-test loop structure nested within a counted loop structure. The corresponding flowchart is as follows.



30.3 Converting Flowcharts to PHP Scripts

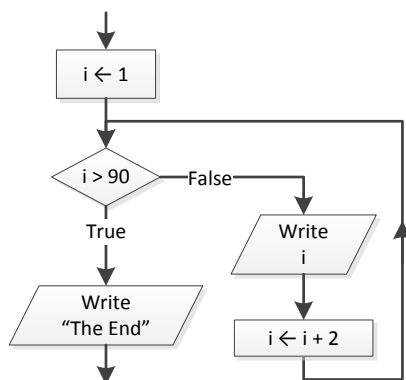
This conversion is not always an easy one. There are cases in which the flowchart designers follow no particular rules, so the initial flowchart may need some modifications before it can become a PHP script. The following is an example of one such case.



As you can see, the loop control structures included in this flowchart fragment match none of the structures that you have already learned, such the pre-test, the post-test, the counted, or even the new one, the mid-test loop structure. Thus, you have only one choice and this is to modify the flowchart by adding extra statements or removing existing ones until known loop control structures start to appear. Following are some exercises in which the initial flowchart does need modification.

Exercise 30.3-1 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart fragment.



This is an easy one. The only obstacle you have to overcome is that the true and false paths are not quite in the right position. You need the true and not the false path to actually iterate. As you already know, it is possible to switch the two paths but you need to negate the Boolean expression as well. Thus, the code fragment becomes

```
$i = 1;
while ($i <= 90) {
    echo $i;
    $i = $i + 2;
}

echo "The End";
```

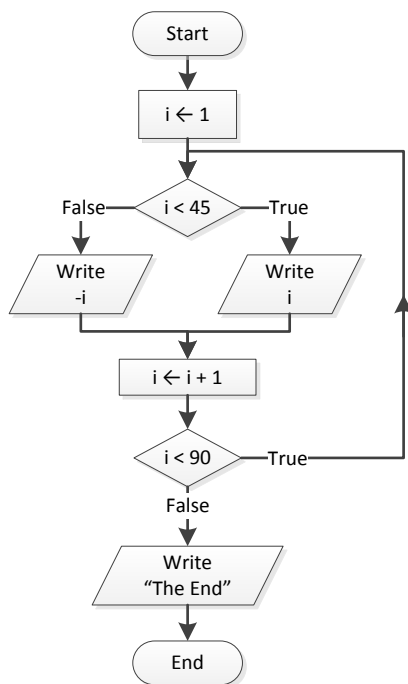
or you can even use a for statement

```
for ($i = 1; $i <= 90; $i += 2) {
    echo $i;
}

echo "The End";
```

Exercise 30.3-2 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart.



Solution

This flowchart contains a post-test loop structure that nests a dual-alternative decision structure. The PHP script is as follows.

```
<?php
```

```
    $i = 1;
```

```
    do {
```

```
        if ($i < 45) {
            echo $i;
        }
```

```
        else {
            echo -$i;
        }
```

```
        $i++;
```

```
    } while ($i < 90);
```

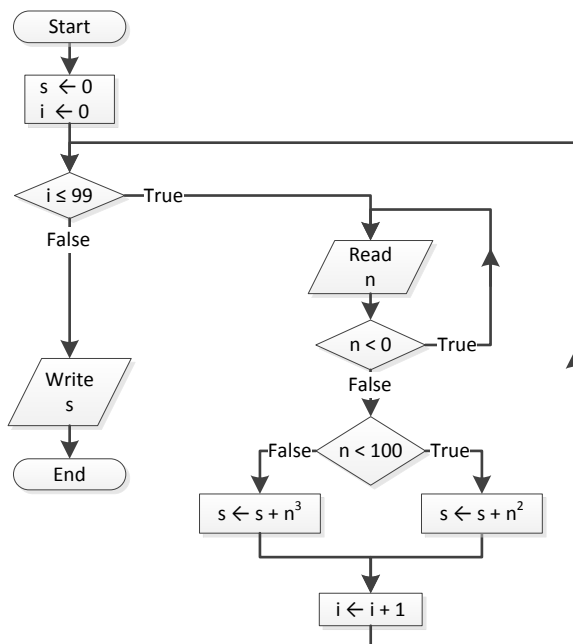
```
    echo "The End";
```

```
?>
```

This is the dual-alternative decision structure

Exercise 30.3-3 Writing the PHP Script

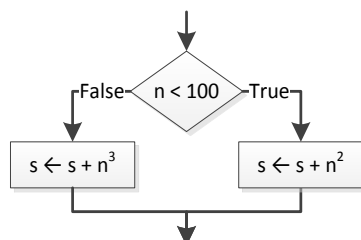
Write the PHP script that corresponds to the following flowchart.



Solution

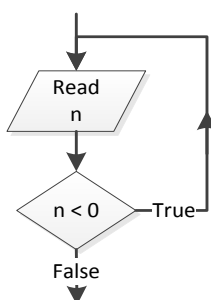
Oops! What a mess! So many diamonds here! Be careful though, because one of them is actually a decision control structure! Yes, you heard it, in this flowchart a decision control structure exists! Can you spot it?

You should be quite familiar with loop control structures so far. As you already know, in loop control structures, one of the diamond's (rhombus's) exits always has an upward direction. Thus, the following flowchart fragment, extracted from the initial one, is obviously the decision control structure that you are looking for.

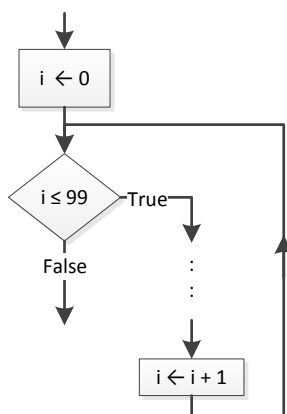


And of course, it's a dual-alternative decision structure!

Now, let's identify the rest of the structures. Right before the dual-alternative decision structure, there is a post-test loop structure. Its flowchart fragment is as follows.



And finally, both the dual-alternative decision structure and the post-test loop structure, mentioned before, are nested within the next flowchart fragment,



which happens to be a pre-test loop structure and can be written in PHP using either a `while` or a `for` statement. The corresponding PHP script is as follows.

```
<?php
```

```
$s = 0;
```

```
for ($i = 0; $i <= 99; $i++) {
```

```
do {
```

```
    $n = trim(fgets(STDIN));
```

```
} while ($n < 0);
```

```
if ($n < 100)
```

```
    $s = $s + pow($n, 2);
```

```
else
```

```
    $s = $s + pow($n, 3);
```

```
}
```

```
echo $s;
```

```
?>
```

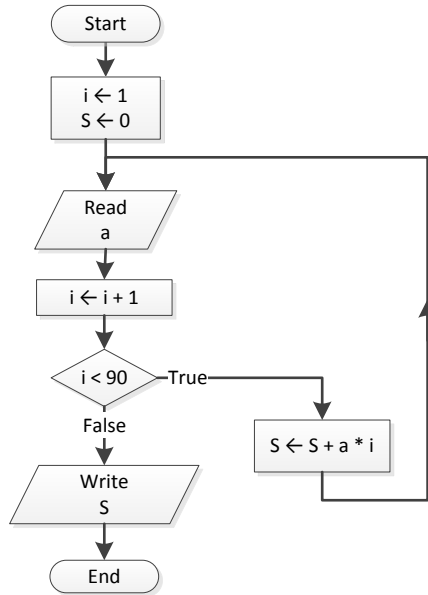
This is the post-test loop structure

This is the dual-alternative decision structure

Wasn't so difficult after all, was it?

Exercise 30.3-4 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart.

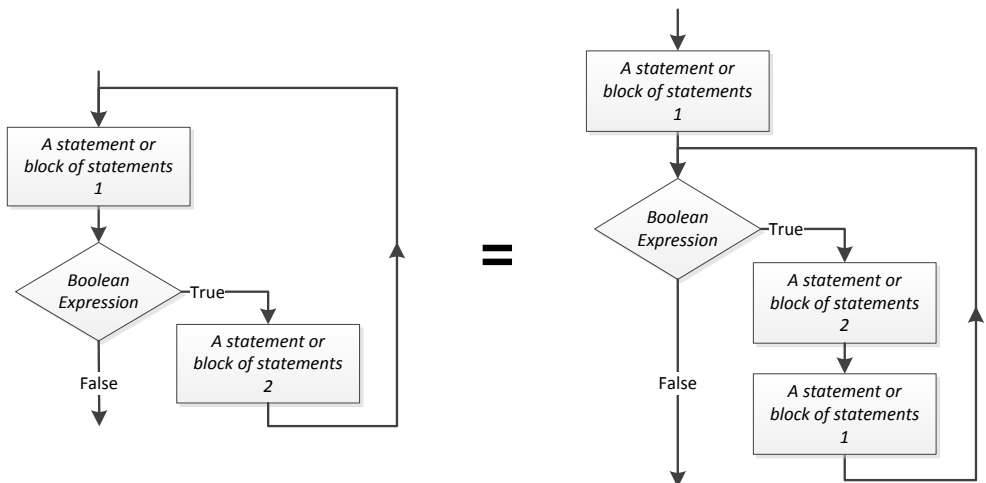


Solution

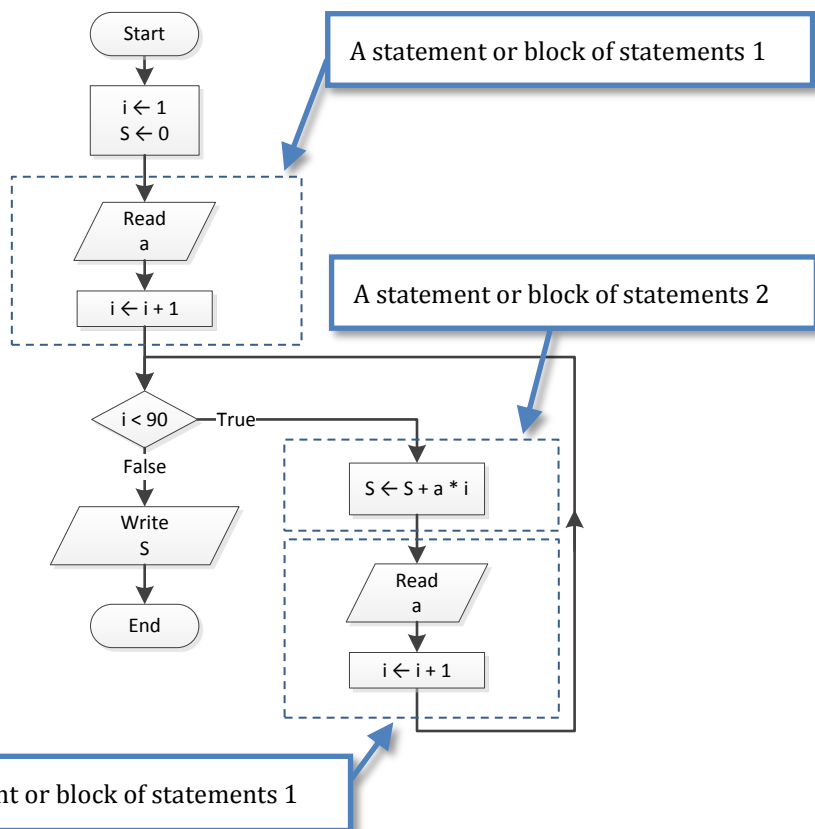
This is the mid-test loop structure mentioned previously in this chapter. Since there is no direct PHP statement for this structure you must convert the flowchart to something more familiar—or you can also use the `break` statement as shown in the next two approaches.

First Approach – Converting the flowchart

For your convenience, the mid-test loop structure and its equivalent, using a pre-test loop structure, are as follows.



Accordingly, the initial flowchart becomes



Now, it's easy to write the corresponding PHP script.

```
<?php
```

```
$i = 1;
```

```
$S = 0;
```

```
$a = trim(fgets(STDIN));
$i++;
```

A statement or block of statements 1

```
while ($i < 90) {
```

```
$S = $S + $a * $i;
```

A statement or block of statements 2

```
$a = trim(fgets(STDIN));
$i++;
```

A statement or block of statements 1

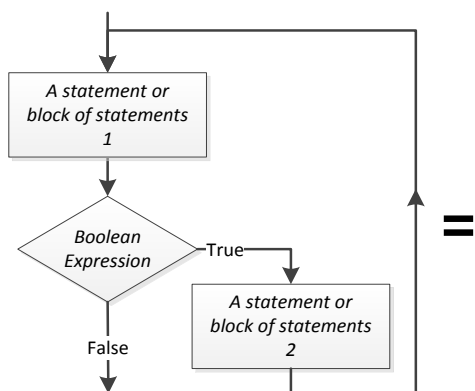
```
}
```

```
echo $S;
```

```
?>
```

Second Approach – Using the break statement

For mid-test loops, there is actually one more approach that uses the `break` statement. The mid-test loop and its PHP script equivalent are as follows.



```

while (true) {
    A statement or block of statements 1
    if !(Boolean_Expression) break;
    A statement or block of statements 2
}
  
```

The main idea is to create an endless loop `while(true) { ... }` and break out of it when the Boolean expression that exists between the two statements or blocks of statements evaluates to `true`.

According to this approach, the initial flowchart can be written directly in PHP as follows, without actually modifying the initial flowchart.

```
<?php
```

```
$i = 1;
```

```
$S = 0;
```

```
while (true) {
```

```
    $a = trim(fgets(STDIN));
    $i++;
```

A statement or block of statements 1

```
    if ($i >= 90) break;
```

```
    $S = $S + $a * $i;
```

A statement or block of statements 2

```
}
```

```
echo $S;
```

```
?>
```

Obviously this is a much better solution! However, keep in mind that even though the `break` statement can sometimes be useful, it may also lead you to write code that is difficult to read and understand, especially when you make extensive use of it. So, please use it cautiously and sparingly!

30.4 Review Exercises

Complete the following exercises.

1. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $i = -100;
    while ($i <= 100) {
        echo $i;
        $i++;
    }
?>
```

2. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $i = trim(fgets(STDIN));
    do {
        echo $i;
        $i++;
    } while ($i <= 100);
?>
```

3. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $a = trim(fgets(STDIN));
    $b = trim(fgets(STDIN));
    for ($i = $a; $i <= $b; $i++) {
        echo $i;
    }
?>
```

4. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $i = 35;
    while ($i > -35) {
        if ($x % 2 == 0)
            echo 2 * $i;
        else
            echo 3 * $i;
        $i--;
    }
?>
```

5. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $i = -20;
    do {
        $x = trim(fgets(STDIN));
        if ($x == 0)
            echo "Zero";
        elseif ($x % 2 == 0)
            echo 2 * $i;
        else
            echo 3 * $i;
        $i++;
    } while ($i <= 20);
?>
```

6. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $a = trim(fgets(STDIN));
    if ($a > 0) {
        $i = 0;
        while ($i <= $a) {
            echo $i;
            $i += 5;
        }
    }
    else {
        echo "Non-Positive Entered!";
    }
?>
```

7. Design the flowchart that corresponds to the following PHP script.

```
<?php
    $a = trim(fgets(STDIN));
    if ($a > 0) {
        $i = 0;
        while ($i <= $a) {
            echo 3 * $i + $i / 2;
            $i++;
        }
    }
    else {
```



```
$i = 10;
do {
    echo 2 * $i - $i / 3;
    $i -= 3;
} while ($i >= $a);
}
?>
```

8. Design the flowchart that corresponds to the following PHP script.

```
<?php
$a = trim(fgets(STDIN));
if ($a > 0) {
    for ($i = 0; $i <= $a; $i++) {
        echo 3 * $i + $i / 2;
    }
}
elseif ($a == 0) {
    $b = trim(fgets(STDIN));
    while ($b > 0) {
        $b = trim(fgets(STDIN));
    }
    echo 2 * $a + $b;
}
else {
    $b = trim(fgets(STDIN));
    while ($b < 0) {
        $b = trim(fgets(STDIN));
    }
    for ($i = $a; $i <= $b; $i++) {
        echo $i;
    }
}
?>
```

9. Design the flowchart that corresponds to the following PHP script.

```
<?php
$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));
$d = trim(fgets(STDIN));
```

```
$sum = 0;
for ($i = $a; $i <= $b - 1; $i++) {
    for ($j = $c; $j <= $d - 1; $j += 2) {
        $sum += $i + $j;
    }
}
echo $sum;
?>
```

10. Design the flowchart that corresponds to the following PHP script.

```
<?php
$s = 0;
for ($i = 1; $i <= 50; $i++) {
    do {
        $n = trim(fgets(STDIN));
    } while ($n < 0);
    $s += sqrt($n);
}
echo $s;
?>
```

11. Design the flowchart that corresponds to the following PHP script.

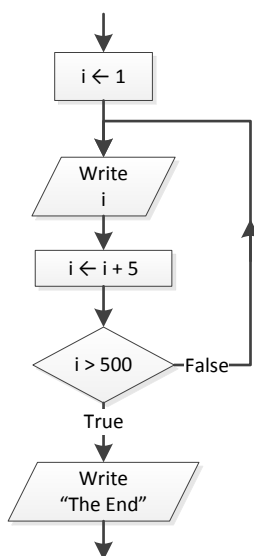
```
<?php
do {
    do {
        $a = trim(fgets(STDIN));
    } while ($a < 0);
    do {
        $b = trim(fgets(STDIN));
    } while ($b < 0);
    echo abs($a - $b);
} while (abs($a - $b) > 100);
?>
```

12. Design the flowchart that corresponds to the following PHP script.

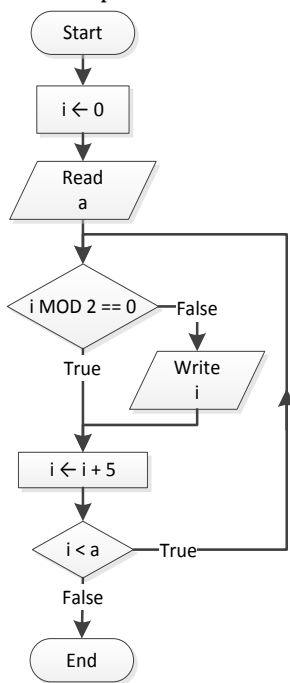
```
<?php
do {
    do {
        $a = trim(fgets(STDIN));
        $b = trim(fgets(STDIN));
    } while ($a < 0 || $b < 0);
```

```
if ($a > $b) {  
    echo $a - $b;  
}  
else {  
    echo $a * $b;  
}  
} while (abs($a - $b) > 100);  
?>
```

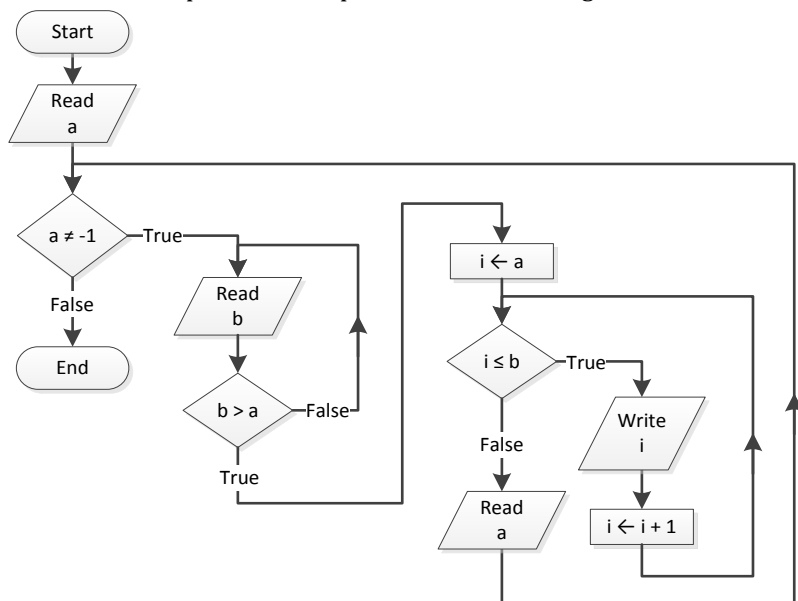
13. Write the PHP script that corresponds to the following flowchart fragment.



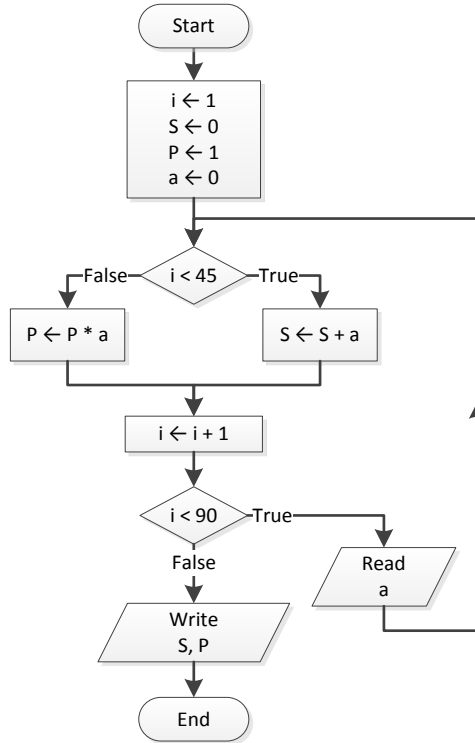
14. Write the PHP script that corresponds to the following flowchart .



15. Write the PHP script that corresponds to the following flowchart.



16. Write the PHP script that corresponds to the following flowchart.



Chapter 31

More Exercises with Loop Control Structures

31.1 Simple Exercises with Loop Control Structures

Exercise 31.1-1 Finding the Sum of 1 + 2 + 3 + ... + 100

Write a PHP script that calculates and displays the following sum:

$$S = 1 + 2 + 3 + \dots + 100$$

Solution

Let's study this exercise using a sequence control structure.

```
$s = 0;
$i = 1;

$s = $s + $i;
$i = $i + 1;

...

$s = $s + $i;
$i = $i + 1;
```

Since the PHP script must calculate the sum of 1 + 2 + 3 + ... + 100, this pair of statements should be written 100 times.

Let's use a trace table to better understand it.

Step	Statement	Notes	\$i	\$s
1	<code>\$s = 0</code>	0	?	0
2	<code>\$i = 1</code>		1	0
3	<code>\$s = \$s + \$i</code>	0 + 1 = 1	1	1
4	<code>\$i = \$i + 1</code>		2	1
5	<code>\$s = \$s + \$i</code>	0 + 1 + 2 = 3	2	3
6	<code>\$i = \$i + 1</code>		3	3
7	<code>\$s = \$s + \$i</code>	0 + 1 + 2 + 3 = 6	3	6
8	<code>\$i = \$i + 1</code>		4	6
...	...		...	...
...	...		...	...
Last but one iteration	<code>\$s = \$s + \$i</code>		99	4950
	<code>\$i = \$i + 1</code>		100	4950

Last iteration	<code>\$s = \$s + \$i</code>	<code>0 + 1 + 2 + 3 + ...+ 99 + 100 = 5050</code>	100	5050
	<code>\$i = \$i + 1</code>		101	5050

Now that everything has been cleared up, you can do the same using instead a counted loop structure that increments variable `$i` from 1 to 100. In each iteration, its value is accumulated in variable `$s`.

file_31.1-1.php

```
<?php
    $s = 0;
    for ($i = 1; $i <= 100; $i++) {
        $s = $s + $i;
    }
    echo $s;
?>
```

Exercise 31.1-2 *Finding the Product of $2 \times 4 \times 6 \times 8 \times 10$*

Write a PHP script that calculates and displays the following product:

$$P = 2 \times 4 \times 6 \times 8 \times 10$$

Solution

Let’s study this exercise using a sequence control structure.

```
$p = 1;
$i = 2;

$p = $p * $i;
$i = $i + 2;

$p = $p * $i;
$i = $i + 2;

$p = $p * $i;
$i = $i + 2;

$p = $p * $i;
$i = $i + 2;
```

Since the script must calculate the product of $2 \times 4 \times 6 \times 8 \times 10$, this pair of statements should be written 5 times.

Let’s use a trace table to better understand it.

Step	Statement	Notes	\$i	\$p
1	\$p = 1	1	?	1
2	\$i = 2		2	1
3	\$p = \$p * \$i	1 × 2 = 2	2	2
4	\$i = \$i + 2		4	2
5	\$p = \$p * \$i	2 × 4 = 8	4	8
6	\$i = \$i + 2		6	8
7	\$p = \$p * \$i	2 × 4 × 6 = 48	6	48
8	\$i = \$i + 2		8	48
9	\$p = \$p * \$i	2 × 4 × 6 × 8 = 384	8	384
10	\$i = \$i + 2		10	384
Last iteration	\$p = \$p * \$i	2 × 4 × 6 × 8 × 10 = 3840	8	3840
	\$i = \$i + 2		12	3840

Now that everything has been cleared up, you can do the same using instead a counted loop structure that increments variable `$i` by 2 from 2 to 10.

file_31.1-2.php

```
<?php
    $p = 1;
    for ($i = 2; $i <= 10; $i += 2) {
        $p = $p * $i;
    }
    echo $p;
?>
```

Exercise 31.1-3 *Finding the Sum of $2^2 + 4^2 + 6^2 + \dots (2N)^2$*

Write a PHP script that lets the user enter an integer *N* and then calculates and displays the following sum:

$$S = 2^2 + 4^2 + 6^2 + \dots (2N)^2$$

Solution

In this exercise, variable `$i` must increment by 2, but it must be raised to the second power before it is accumulated in variable `$s`.

The final PHP script is as follows.

file_31.1-3.php

```
<?php
```

```

$N = trim(fgets(STDIN));
$s = 0;
for ($i = 2; $i <= 2 * $N; $i += 2) {
    $s = $s + pow($i, 2);
}

echo $s;
?>

```

Exercise 31.1-4 *Finding the Sum of $3^3 + 6^6 + 9^9 + \dots (3N)^{3N}$*

Write a PHP script that lets the user enter an integer N and then calculates and displays the following sum:

$$S = 3^3 + 6^6 + 9^9 + \dots + (3N)^{3N}$$

Solution

This is pretty much the same as the previous exercise. The only difference is that variable $\$i$ must be raised to the i^{th} power before it is accumulated in variable $\$s$. The final PHP script is as follows.

file_31.1-4.php

```

<?php
$N = trim(fgets(STDIN));
$s = 0;
for ($i = 3; $i <= 3 * $N; $i += 3) {
    $s = $s + pow($i, $i);
}
echo $s;
?>

```

Exercise 31.1-5 *Finding the Average Value of Positive Numbers*

Write a PHP script that lets the user enter 100 numbers and then calculates and displays the average value of the positive numbers. Add all necessary checks to make the script satisfy the property of definiteness.

Solution

Since the total number of iterations is known, you can use a counted loop structure. Inside the loop, however, a decision control structure must check whether or not the given number is positive; if so, it must accumulate the given number in variable $\$s$. When the flow of execution exits the loop, the average value can then be calculated. The PHP script is as follows.

file_31.1-5.php

```
<?php
    $s = 0;
    $count = 0;
    for ($i = 1; $i <= 100; $i++) {
        $x = trim(fgets(STDIN));
        if ($x > 0) {
            $s = $s + $x;
            $count++;
        }
    }
    if ($count != 0) {
        echo $s / $count;
    }
    else {
        echo "No numbers entered!";
    }
?>
```

Notice: The `if ($count != 0)` statement is necessary, because there is a possibility that the user may enter negative values only. By including this check, the script prevents any division-by-zero errors and thereby satisfies the property of definiteness.

Exercise 31.1-6 Counting the Numbers According to Which is Greater

Write a PHP script that prompts the user to enter 10 pairs of numbers and then counts and displays the number of times that the first number given was greater than the second one.

Solution

Once again, a counted loop structure can be used. The PHP script is as follows.

file_31.1-6.php

```
<?php
    $count_a = 0;
    $count_b = 0;

    for ($i = 1; $i <= 10; $i++) {
        echo "Enter number A: ";
        $a = trim(fgets(STDIN));
        echo "Enter number B: ";
        $b = trim(fgets(STDIN));
```

```
if ($a > $b) {
    $count_a++;
}
elseif ($b > $a) {
    $count_b++;
}
}
echo $count_a, $count_b;
?>
```

A reasonable question that someone may ask is “*Why is a multiple-decision control structure being used? Why not use a dual-alternative decision structure instead?*”

Suppose that a dual-alternative decision structure, such as the following, is used.

```
if ($a > $b) {
    $count_a++;
}
else {
    $count_b++;
}
```

In this decision control structure, the variable `$count_b` increments when variable `$b` is greater than variable `$a` (this is desirable) but also when variable `$b` is equal to variable `$a` (this is undesirable). Using a multiple-decision control structure instead ensures that variable `$count_b` increments only when variable `$b` is greater than (and not when it is equal to) variable `$a`.

Exercise 31.1-7 *Counting the Numbers According to Their Digits*

Write a PHP script that prompts the user to enter 20 integers and then counts and displays the total number of one-digit, two-digit, and three-digit integers. Assume that the user enters values between 1 and 999.

Solution

Nothing new here! The PHP script is as follows.

file_31.1-7.php

```
<?php
$count1 = 0;
$count2 = 0;
$count3 = 0;

for ($i = 1; $i <= 20; $i++) {
    echo "Enter a number: ";
```

```
$a = trim(fgets(STDIN));

if ($a <= 9) {
    $count1++;
}
elseif ($a <= 99) {
    $count2++;
}
else {
    $count3++;
}
}
echo $count1, $count2, $count3;

?>
```

Exercise 31.1-8 *How Many Numbers Fit in a Sum*

Write a PHP script that lets the user enter numeric values repeatedly until the sum of them exceeds 1000. At the end, the script should display the total quantity of numbers entered.

Solution

In this case, you don't know the exact number of iterations, so you cannot use a counted loop structure. Let's use a post-test loop structure instead, but, in order to make your script free of logic errors you should follow the "Ultimate" rule discussed in Chapter 29. According to this rule, the post-test loop structure should always be as follows, given in general form.

```
initialize $sum
do {
    A statement or block of statements
    Update/alter $sum
} while ( Boolean_Expression($sum) )
```

Following this, the PHP script becomes

file_31.1-8.php

```
<?php
$count = 0;
$sum = 0;                                //Initialization of $sum
do {
    $x = trim(fgets(STDIN));
    $count++;
    $sum += $x;                          //Update/alteration of $sum
```

```
} while ($sum <= 1000);  
echo $count;  
?>
```

Exercise 31.1-9 Finding the Sum of Integers

Write a PHP script that prompts the user to enter integer values repeatedly until a real one is entered. At the end, the script should display the total number of positive integers entered.

Solution

Once again, you don't know the exact number of iterations, so you cannot use a counted loop structure.

According to the “Ultimate” rule, the pre-test loop structure should always be as follows, given in general form.

```
$x = trim(fgets(STDIN)); //Initialization of $x  
while ( intval($x) == $x ) {  
    A statement or block of statements  
    $x = trim(fgets(STDIN)); //Update/alteration of $x  
}
```

Notice: In this case, using a pre-test loop structure rather than a post-test loop structure is a better option. Why? Because if the first number given is real, the pre-test loop structure ensures that the flow of execution never enters the loop!

The final PHP script is as follows.

file_31.1-9.php

```
<?php  
$count = 0;  
  
echo "Enter a number: ";  
$x = trim(fgets(STDIN));  
while (intval($x) == $x) {  
    if ($x > 0) {  
        $count++;  
    }  
    echo "Enter a number: ";  
    $x = trim(fgets(STDIN));  
}  
echo $count;  
?>
```

Exercise 31.1-10 *Iterating as Many Times as the User Wishes*

Write a PHP script that prompts the user to enter two numbers and then calculates and displays the first number raised to the power of the second one. The script should iterate as many times as the user wishes. At the end of each area calculation, the script should ask the user if he or she wishes to calculate the area of another rectangle. If the answer is “yes” the script should repeat; it should end otherwise. Make your script accept the answer in all possible forms such as “yes”, “YES”, “Yes”, or even “YeS”.

Solution

According to the “Ultimate” rule, the pre-test loop structure should be as follows, given in general form.

```
$answer = "yes"; //Initialization of $answer
while (strtoupper($answer) == "YES") {
    //Here goes the code that
    //prompts the user to enter two numbers and then
    //calculates and displays the first number
    //raised to the power of the second one.

    echo "Would you like to repeat?";
    $answer = trim(fgets(STDIN)); //Update/alteration of $answer
}
```

The `strtoupper()` function ensures that the script operates properly for any given answer: “yes”, “YES”, “Yes”, or even “YeS” or “yEs”!

On the other hand, if you decide to use a post-test loop structure, it should be as follows, given in general form.

```
$answer = "YES"; //Redundant initialization of $answer
do {
    //Here goes the code that
    //prompts the user to enter two numbers and then
    //calculates and displays the first number
    //raised to the power of the second one.

    echo "Would you like to repeat?";
    $answer = trim(fgets(STDIN)); //Update/alteration of $answer
} while (strtoupper($answer) == "YES");
```

Using a post-test loop is a better approach since the initialization of `$answer` is actually redundant and can be omitted. Why? In contrast to the pre-test loop structure, the flow of execution enters the loop either way and initialization of `$answer` occurs inside the loop.

Accordingly, the solution to this exercise, using a post-test loop structure becomes

file_31.1-10.php

```
<?php
do {
    echo "Enter two numbers: ";
    $a = trim(fgets(STDIN));
    $b = trim(fgets(STDIN));

    $result = pow($a, $b);
    echo "The result is: ", $result, "\n";

    echo "Would you like to repeat?";
    $answer = trim(fgets(STDIN));
} while (strtoupper($answer) == "YES");
?>
```

Exercise 31.1-11 *Finding the Sum of the Digits*

Write a PHP script that lets the user enter an integer and then calculates the sum of its digits. Afterwards, use the value 4753 as an input value and create a trace table to determine the values of the variables in each step.

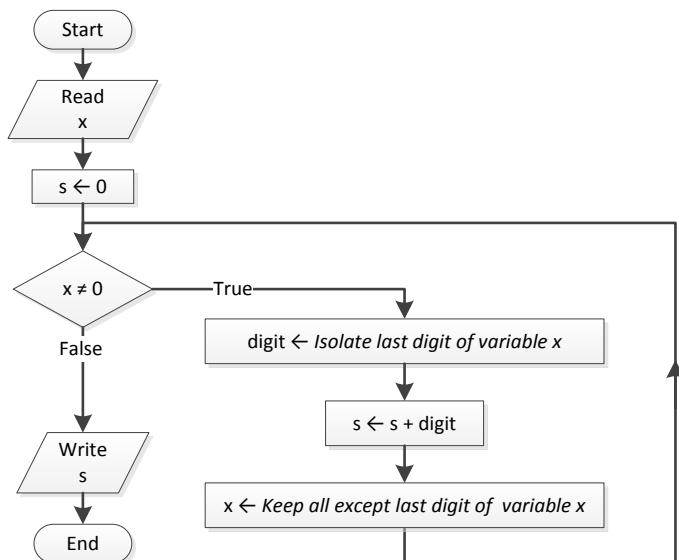
Solution

In Chapter 13, you learned how to split the digits of an integer when its total number of digits was known. In this exercise however, the user is allowed to enter any value, no matter how small or large; thus, the total number of the digits can be unknown.

To solve this exercise, a loop control structure could be used. The main idea is to isolate one digit at each iteration. But what you don't really know is the total number of iterations that should be performed because this number depends on the given integer. So, is this a dead end? Of course not!

Inside the loop, the given integer can be getting “smaller” and “smaller” in every iteration until eventually it becomes zero. The value of zero can serve to stop the loop control structure from iterating. For example, if the given number is 4753, it can become 475 in the first iteration, then 47, then 4, and finally 0. When it becomes 0, the iterations can stop.

Let's try to comprehend the proposed solution using the following flowchart. Some statements are written in general form.



The statement

$\text{digit} \leftarrow \text{Isolate last digit of variable } x.$

can be written using the well-known MOD 10 operation:

```
digit ← x MOD 10
```

Of course, the whole concept relies on the statement

$x \leftarrow \text{Keep all except last digit of variable } x.$

This is the statement that eventually zeros the value of variable x , and the flow of execution then exits the loop. To write this statement you can use a DIV 10 operation.

```
x ← x DIV 10
```

Accordingly, the PHP script becomes

file_31.1-11.php

```
<?php
```

```
$x = trim(fgets(STDIN));
```

```
$s = 0;
```

```
while ($x != 0) {
```

```
    $digit = $x % 10;           //This is the x MOD 10 operation
```

```
    $s = $s + $digit;
```

```
    $x = intval($x / 10);       //This is the x DIV 10 operation
```

```
}
```

```
echo $s;
```

```
?>
```

Now let’s create a trace table for the input value 4753.

Step	Statement	Notes	\$x	\$digit	\$s
1	<code>\$x = trim(fgets(STDIN));</code>	User enters the value 4753	4753	?	?
2	<code>\$s = 0</code>		4753	?	0
3	<code>while (\$x != 0)</code>	This validates to true			
4	<code>\$digit = \$x % 10</code>		4753	3	0
5	<code>\$s = \$s + \$digit</code>		4753	3	3
6	<code>\$x = intval(\$x / 10)</code>		475	3	3
7	<code>while (\$x != 0)</code>	This validates to true			
8	<code>\$digit = \$x % 10</code>		475	5	3
9	<code>\$s = \$s + \$digit</code>		475	5	8
10	<code>\$x = intval(\$x / 10)</code>		47	5	8
11	<code>while (\$x != 0)</code>	This validates to true			
12	<code>\$digit = \$x % 10</code>		47	7	8
13	<code>\$s = \$s + \$digit</code>		47	7	15
14	<code>\$x = intval(\$x / 10)</code>		4	7	15
15	<code>while (\$x != 0)</code>	This validates to true			
16	<code>\$digit = \$x % 10</code>		4	4	15
17	<code>\$s = \$s + \$digit</code>		4	4	19
18	<code>\$x = intval(\$x / 10)</code>		0	4	19
19	<code>while (\$x != 0)</code>	This validates to false			
20	<code>echo \$s</code>	The value 19 is displayed			

Exercise 31.1-12 Counting the Digits

Write a PHP script that prompts the user to enter an integer and then displays the number of its digits.

Solution

Using the method proposed in the previous exercise, the PHP script is as follows.

```
file_31.1-12.php
<?php
```

```
echo "Enter a number: ";
$x = trim(fgets(STDIN));
$count = 0;

while ($x != 0) {
    $count++;
    $x = intval($x / 10);
}
echo $count;

?>
```

31.2 Exercises with Nested Loop Control Structures

Exercise 31.2-1 *Displaying all Three-Digit Integers that Contain a Given Digit*

Write a PHP script that prompts the user to enter a digit (0 to 9) and then displays all three-digit integers that contain that given digit at least once. For example, for the given value 7, the values 357, 771, and 700 are such integers.

Solution

There are two different approaches, actually! The first one uses just one counted loop structure while the second one uses three counted loop structures, nested one within the other. Let's analyze them both!

First Approach – Using a counted loop structure and a decision control structure

The main idea is to use a counted loop structure where the *counter* variable increments from 100 to 999. Inside the loop, the *counter* variable is split into its individual digits (digit₃, digit₂, digit₁) and a decision control structure is used to check if at least one of its digits is equal to the given one. The PHP script is as follows.

file_31.2-1a.php

```
<?php
echo "Enter a digit 0 - 9: ";
$x = trim(fgets(STDIN));

for ($i = 100; $i <= 999; $i++) {
    $digit3 = intval($i / 100);
    $r = $i % 100;

    $digit2 = intval($r / 10);
    $digit1 = $r % 10;
```

```
if ($digit3 == $x || $digit2 == $x || $digit1 == $x) {  
    echo $i, "\n";  
}  
}  
?>
```

Second Approach – Using nested loop control structures and a decision control structure

The main idea is to use three counted loop structures, nested one within the other. In this case, there are three *counter* variables and each one of them corresponds to one digit of the three-digit integer. The PHP script is as follows.

file_31.2-1b.php

```
<?php  
echo "Enter a digit 0 - 9: ";  
$x = trim(fgets(STDIN));  
  
for ($digit3 = 1; $digit3 <= 9; $digit3++) {  
    for ($digit2 = 0; $digit2 <= 9; $digit2++) {  
        for ($digit1 = 0; $digit1 <= 9; $digit1++) {  
            if ($digit3 == $x || $digit2 == $x || $digit1 == $x) {  
                echo $digit3 * 100 + $digit2 * 10 + $digit1, "\n";  
            }  
        }  
    }  
}  
?  
?>
```

If you follow the flow of execution, the value 100 is the first “integer” evaluated ($\$digit1 = 1, \$digit2 = 0, \$digit3 = 0$). Then, the most-nested loop control structure increments variable $\$digit1$ by one and the next value evaluated is “integer” 101. This continues until $\$digit1$ reaches the value 9; that is, until the “integer” reaches the value 109. The flow of execution then exits the most-nested loop control structure, variable $\$digit2$ increments by one, and the most-nested loop control structure starts over again, thus the values evaluated are the “integers” 110, 111, 112, ... 119. The process goes on until all integers up to the value 999 are evaluated.

Notice: Please note that variable $\$digit3$ starts from 1, whereas variables $\$digit2$ and $\$digit1$ start from 0. This is necessary since the scale for three-digit numbers begins from 100 and not from 000.

Notice: Please note how the echo statement composes the three-digit integer.

Exercise 31.2-2 *Displaying all Instances of a Specified Condition*

Write a PHP script that displays all three-digit integers in which the first digit is smaller than the second digit and the second digit is smaller than the third digit. For example, the values 357, 456, and 159 are such integers.

Solution

There are three different approaches, actually! Let's analyze them all!

First Approach – Using a counted loop structure and a decision control structure

The PHP script is as follows.

file_31.2-2a.php

```
<?php
for ($i = 100; $i <= 999; $i++) {
    $digit3 = intval($i / 100);
    $r = $i % 100;

    $digit2 = intval($r / 10);
    $digit1 = $r % 10;

    if ($digit3 < $digit2 && $digit2 < $digit1) {
        echo $i, "\n";
    }
}
?>
```

Second Approach – Using nested loop control structures and a decision control structure

The PHP script is as follows.

file_31.2-2b.php

```
<?php
for ($digit3 = 1; $digit3 <= 9; $digit3++) {
    for ($digit2 = 0; $digit2 <= 9; $digit2++) {
        for ($digit1 = 0; $digit1 <= 9; $digit1++) {
            if ($digit3 < $digit2 && $digit2 < $digit1) {
                echo $digit3 * 100 + $digit2 * 10 + $digit1, "\n";
            }
        }
    }
}
?>
```

Third Approach – Using nested loop control structures only

This approach is based on the previous one. The main difference between them is that in this case, variable `$digit1` always begins from a value greater than `$digit2`, and variable `$digit2` always begins from a value greater than `$digit3`. In that way, the first integer that will be displayed is 123.

Notice: *There are no integers below the value 123 and above the value 789 that can validate the Boolean expression `$digit3 < $digit2 && $digit2 < $digit1` to true.*

The PHP script is as follows.

file_31.2-2c.php

```
<?php
for ($digit3 = 1; $digit3 <= 7; $digit3++) {
    for ($digit2 = $digit3 + 1; $digit2 <= 8; $digit2++) {
        for ($digit1 = $digit2 + 1; $digit1 <= 9; $digit1++) {
            echo $digit3 * 100 + $digit2 * 10 + $digit1, "\n";
        }
    }
}
?>
```

Notice: *This solution is the most efficient since it doesn't use any decision control structure and, moreover, the number of iterations is kept to a minimum!*

Remember! *As you can see, one problem can have many solutions. It is up to you to find the optimal one!*

31.3 Data Validation with Loop Control Structures

Data validation is the process of restricting data input, forcing the user to enter only valid values.

You have already encountered one method of data validation using decision control structures. Let's recall a previous example.

```
<?php
echo "Enter a number: ";
$x = trim(fgets(STDIN));

if (is_numeric($x) == true) {
    if ($x > 0) {
        echo $x, " is positive";
    }
}
else {
    echo "Invalid Number";
}
```

```
}  
?>
```

The statement `if (is_numeric($x) == true)` checks if the user has entered a valid number and if not, an error message is displayed.

This method, however, is not the most appropriate because if the user does enter an invalid number, the script displays the error message and the flow of execution inevitably reaches the end. The user must then execute the script again in order to re-enter a valid number.

Next, you will see three approaches given in general form that validate data input using loop control structures. If the user persistently enters an invalid value, the main idea is to prompt him or her repeatedly, until he or she gets bored and finally enters a valid one. Of course, if the user enters a valid value right from the beginning, the flow of execution simply continues to the next part of the script.

Which approach you use depends on whether or not you wish to display an error message and whether you wish to display an individual error message for each error or just a single error message for any kind of error.

First Approach – Validating data input without error messages

To validate data input without displaying any error messages, you can use the following code fragment given in general form.

```
do {  
    echo "Prompt message";  
    $input_data = trim(fgets(STDIN));  
} while ($input_data test 1 fails || $input_data test 2 fails || ...);
```

Second Approach – Validating data input with one error message

To validate data input and display an error message (that is, the same error message for any type of input error), you can use the following code fragment given in general form.

```
echo "Prompt message";  
$input_data = trim(fgets(STDIN));  
while ($input_data test 1 fails || $input_data test 2 fails || ... ) {  
    echo "Error message \n";  
    echo "Prompt message";  
    $input_data = trim(fgets(STDIN));  
}
```

Third Approach – Validating data input with individual error messages

To validate data input and display an error message (that is, individual error messages, one for each type of input error), you can use the following code fragment given in general form.

```
do {  
    echo "Prompt message";
```

```
$input_data = trim(fgets(STDIN));  
$failure = false;  
if ($input_data test 1 fails) {  
    echo "Error message 1 \n";  
    $failure = true;  
}  
elseif ($input_data test 2 fails) {  
    echo "Error message 2 \n";  
    $failure = true;  
}  
elseif (...  
    ...  
}  
} while ($failure == true);
```

Exercise 31.3-1 *Finding the Square Root - Validation Without Error Messages*

Write a PHP script that prompts the user to enter a numeric value and then calculates its square root. Moreover, the script should validate data input, allowing the user to enter only non-negative¹ numeric values. There is no need to display any error messages.

Solution

First, let's write the script without data validation.

```
<?php
```

```
echo "Enter a number: ";  
$x = trim(fgets(STDIN));
```

Data input stage without validation.

```
$y = sqrt($x);  
echo $y;
```

```
?>
```

To validate data input without displaying any error messages you can use the first approach. All you need to do is replace the statements marked with a dashed rectangle with the following code fragment.

```
do {  
    echo "Enter a non-negative number: ";  
    $x = trim(fgets(STDIN));  
} while (is_numeric($x) != true || $x < 0);
```

¹ A quantity that is either zero or positive.

The final PHP script becomes

file_31.3-1.php

```
<?php
do {
    echo "Enter a non-negative number: ";
    $x = trim(fgets(STDIN));
} while (is_numeric($x) != true || $x < 0);

$y = sqrt($x);
echo $y;
?>
```

Data input validation
without error messages.

Exercise 31.3-2 *Finding the Square Root - Validation with One Error Message*

Write a PHP script that prompts the user to enter a numeric value and then calculates its square root. Moreover, the script should validate data input and display an error message when the user enters any non-numeric or negative values.

Solution

In the previous exercise, the two statements marked with the dashed rectangle were replaced with a code fragment that validated data input without displaying any error message. The same method is followed here, except that the replacing PHP fragment is based on the second approach. The PHP script is as follows.

file_31.3-2.php

```
<?php
echo "Enter a non-negative number: ";
$x = trim(fgets(STDIN));
while (is_numeric($x) != true || $x < 0) {
    echo "Error: Invalid number! \n";
    echo "Enter a non-negative number: ";
    $x = trim(fgets(STDIN));
}

$y = sqrt($x);
echo $y;
?>
```

Data input validation with
one single error message.

Exercise 31.3-3 *Finding the Square Root - Validation with Individual Error Messages*

Write a PHP script that prompts the user to enter a numeric value and then calculates its square root. Moreover, the script should validate data input and display individual error messages when the user enters any non-numeric or negative values.

Solution

To validate data input and display individual error messages for each error, you can use the third approach. The PHP script is as follows.

file_31.3-3.php

```
<?php
do {
    echo "Enter a non-negative number: ";
    $x = trim(fgets(STDIN));

    $failure = false;
    if (is_numeric($x) != true) {
        echo "Please enter numeric values! \n";
        $failure = true;
    }
    elseif ($x < 0) {
        echo "Please enter non-negative numbers! \n";
        $failure = true;
    }
} while ($failure == true);

$y = sqrt($x);
echo $y;
?>
```

Data input validation with an individual error message for each error.

Exercise 31.3-4 *Finding the Sum of 10 Numbers*

Write a PHP script that prompts the user to enter 10 numbers and then calculates and displays their sum. Moreover, the script should validate data input and display an error message when the user enters any non-numeric or negative values.

Solution

This exercise was already discussed in Chapter 27. The only difference here is that this script should validate data input and display an error message when the user enters invalid values. For your convenience, the solution proposed in Chapter 27 is reproduced next.

<?php

`$sum = 0;``for ($i = 1; $i <= 10; $i++) {` `echo "Enter a number: ";` `$a = trim(fgets(STDIN));` `$sum = $sum + $a;``}``echo $sum;``?>`

Data input stage without validation

Now, replace the statements marked with a dashed rectangle with the following code fragment

`echo "Enter a number: ";``$a = trim(fgets(STDIN));``while (is_numeric($a) != true) {` `echo "Please enter numeric values! \n";` `echo "Enter a number: ";` `$a = trim(fgets(STDIN));`

and the final PHP script becomes

file_31.3-4.php

<?php

`$sum = 0;``for ($i = 1; $i <= 10; $i++) {` `echo "Enter a number: ";` `$a = trim(fgets(STDIN));` `while (is_numeric($a) != true) {` `echo "Please enter numeric values! \n";` `echo "Enter a number: ";` `$a = trim(fgets(STDIN));` `}` `$sum = $sum + $a;``}``echo $sum;``?>`

Data input validation with one single error message

Notice: The basic purpose of this exercise was to show you how to nest the loop control structure that validates data input into other already existing loop control structures.

31.4 Using Loop Control Structures to Solve Mathematical Problems

Exercise 31.4-1 *Calculating the Area of as Many Triangles as the User Wishes*

Write a PHP script that prompts the user to enter the lengths of all three sides A , B , and C of a triangle and then calculates and displays its area. You can use Heron's formula,

$$\text{Area} = \sqrt{S(S-A)(S-B)(S-C)}$$

where S is the semi-perimeter

$$S = \frac{A+B+C}{2}$$

The script should iterate as many times as the user wishes. At the end of each area calculation, the script should ask the user if he or she wishes to calculate the area of another triangle. If the answer is "yes" the script should repeat; it should end otherwise. Make your script accept the answer in all possible forms such as "yes", "YES", "Yes", or even "YeS".

Moreover, the script should validate data input and display an error message when the user enters any non-positive value.

Solution

According to the "Ultimate" rule, the post-test loop structure should be as follows, given in general form.

```
$answer = "yes"; //Redundant initialization of $answer.
do {
    //Here goes the code that prompts the user to enter
    //the lengths of all three sides A, B, C of a triangle and
    //then calculates and displays its area

    echo "Would you like to repeat? ";
    $answer = trim(fgets(STDIN)); //Update/alteration of $answer
} while (strtoupper($answer) == "YES");
```

Remember! The `strtoupper()` function ensures that the scripts operates properly for any given answer "Yes", "yes", "YES" or even "YeS" or "yEs"!

The solution to this exercise, using a post-test loop structure, is as follows.

file_31.4-1.php

```
<?php
do {
    echo "Enter side A: ";
    $a = trim(fgets(STDIN));
```

```

while ($a <= 0) {
    echo "Invalid side. Enter side A: ";
    $a = trim(fgets(STDIN));
}

echo "Enter side B: ";
$b = trim(fgets(STDIN));
while ($b <= 0) {
    echo "Invalid side. Enter side B: ";
    $b = trim(fgets(STDIN));
}

echo "Enter side C: ";
$c = trim(fgets(STDIN));
while ($c <= 0) {
    echo "Invalid side. Enter side C: ";
    $c = trim(fgets(STDIN));
}

$s = ($a + $b + $c) / 2;
$area = sqrt($s * ($s - $a) * ($s - $b) * ($s - $c));
echo "The area is: ", $area, "\n";

echo "Would you like to repeat? ";
$answer = trim(fgets(STDIN));
} while (strtoupper($answer) == "YES");
?>

```

Exercise 31.4-2 *Finding x and y*

Write a PHP script that displays all possible integer values of x and y within the range -20 to +20 that validate the following formula:

$$3x^2 - 6y^2 = 6$$

Solution

If you just want to display all possible combinations of variables x and y , you can use the following PHP script.

```

<?php
for ($x = -20; $x <= 20; $x++) {
    for ($y = -20; $y <= 20; $y++) {

```

```

        echo $x, " ", $y, "\n";
    }
}
?>

```

However, from all those combinations, you need only those that validate the expression $3x^2 - 6y^2 = 6$. A decision control structure is perfect for that purpose! The final PHP script is as follows.

file_31.4-2.php

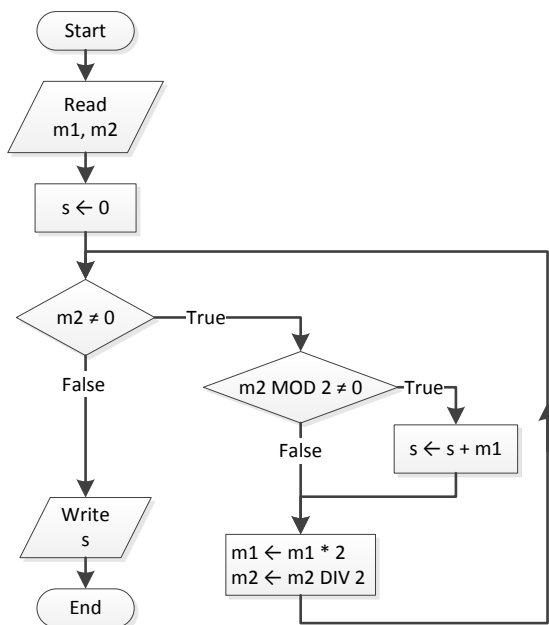
```

<?php
for ($x = -20; $x <= 20; $x++) {
    for ($y = -20; $y <= 20; $y++) {
        if (3 * pow($x, 2) - 6 * pow($y, 2) == 6) {
            echo $x, " ", $y, "\n";
        }
    }
}
?>

```

Exercise 31.4-3 From Russia with Love

You can multiply two positive integers using the “Russian multiplication algorithm,” which is presented in the following flowchart.



Write the corresponding PHP script and create a trace table to determine the values of the variables in each step for the input values 5 and 13.

Solution

In the given flowchart, a single-alternative decision structure is nested within a pre-test loop structure. The corresponding PHP script is as follows.

file_31.4-3a.php

```
<?php
    $m1 = trim(fgets(STDIN));
    $m2 = trim(fgets(STDIN));

    $s = 0;
    while ($m2 != 0) {
        if ($m2 % 2 != 0) {
            $s += $m1;
        }
        $m1 *= 2;
        $m2 = intval($m2 / 2);
    }
    echo $s;
?>
```

For the input values of 5 and 13, the trace table looks like this.

Step	Statement	Notes	\$m1	\$m2	\$s
1	\$m1 = trim(fgets(STDIN))	User enters the value 5	5	?	?
2	\$m2 = trim(fgets(STDIN))	User enters the value 13	5	13	?
3	\$s = 0		5	13	0
4	while (\$m2 != 0)	This evaluates to true			
5	if (\$m2 % 2 != 0)	This evaluates to true			
6	\$s += \$m1		5	13	5
7	\$m1 *= 2		10	13	5
8	\$m2 = intval(\$m2 / 2)		10	6	5
9	while (\$m2 != 0)	This evaluates to true			
10	if (\$m2 % 2 != 0)	This evaluates to false			
11	\$m1 *= 2;		20	6	5
12	\$m2 = intval(\$m2 / 2)		20	3	5

13	while (\$m2 != 0)	This evaluates to true		
14	if (\$m2 % 2 != 0)	This evaluates to true		
15	\$s += \$m1;	20	3	25
16	\$m1 *= 2	40	3	25
17	\$m2 = intval(\$m2 / 2)	40	1	25
18	while (\$m2 != 0)	This evaluates to true		
19	if (\$m2 % 2 != 0)	This evaluates to true		
20	\$s += \$m1	40	1	65
21	\$m1 *= 2	80	1	65
22	\$m2 = intval(\$m2 / 2)	80	0	65
23	while (\$m2 != 0)	This evaluates to false		
24	echo \$s	The value 65 is displayed which is, of course, the result of the multiplication 5 × 13		

However, one thing that you need to know about CPUs and microprocessors is that the only arithmetic operation that most of them can actually perform is addition. The majority of them perform all multiplication operations using the *Russian multiplication algorithm*.

On the other hand, in the program, there are the following two statements:

```
$m1 *= 2;  
$m2 = intval($m2 / 2);
```

But since CPUs can’t multiply or divide, how on Earth are these operations actually performed?

Things are not always as they seem! In this case, these two operations are not performed by multiplying or by dividing by 2, but by shifting binary numbers to the left or right correspondingly.

Remember! All numbers are stored in main memory (RAM) and manipulated by the Central Processing Unit (CPU) in their binary form; that is, in a form that uses zeros and ones. For example, the value 39 is stored in main memory as 00100111.

For example, the value 43 can be represented in binary form as 00101011 which

- after shifting it to the left, becomes 01010110. This is the value 86 in decimal form: (43 × 2 = 86).
- after shifting it to the right, becomes 00010101. This is the value 21 in decimal form: (43 DIV 2 = 21).

Remember! Shifting a binary number to the left is equivalent to multiplying it by 2, and shifting it to the right is equivalent to dividing it by 2 (integer division).

Notice: After shifting a binary number to the right, the rightmost digit is lost.

PHP incorporates two special operators that have purposely not been mentioned so far. Actually, only advanced programmers use them. But since day by day you are becoming a more experienced programmer, it is worth mentioning them here. These operators are called *shifting operators* and they do the obvious: they shift an integer (the binary representation of it) to the left or to the right. The Russian multiplication algorithm is presented here once again, but without any division or multiplication operations, just simple shifts to the left and to the right!

file_31.4-3b.php

```
<?php
$m1 = trim(fgets(STDIN));
$m2 = trim(fgets(STDIN));

$s = 0;
while ($m2 != 0) {
    if ($m2 % 2 != 0) {
        $s += $m1;
    }
    $m1 = $m1 << 1; //This is equivalent to $m1 *= 2.
                    //It shifts the bits of $m1 one position to the left.

    $m2 = $m2 >> 1; //This is equivalent to $m2 = intval($m2 / 2).
                    //It shifts the bits of $m2 one position to the right.
}
echo $s;
?>
```

Exercise 31.4-4 Finding the Number of Divisors

Write a PHP script that lets the user enter a positive integer and then displays the total number of its divisors.

Solution

Let's see some examples.

- The divisors of value 12 are numbers 1, 2, 3, 4, 6, 12.
- The divisors of value 15 are numbers 1, 3, 5, 15.
- The divisors of value 20 are numbers 1, 2, 4, 5, 10, 20.
- The divisors of value 50 are numbers 1, 2, 5, 10, 25, 50.

If variable $\$x$ contains the given integer, all possible divisors of $\$x$ are between 1 and $\$x$. Thus, all you need here is a counted loop structure that increments variable

counter from 1 to $\$x$ and checks, in each iteration, whether the value of *counter* is a divisor of $\$x$. The PHP script is as follows.

file_31.4-4a.php

```
<?php
    $x = trim(fgets(STDIN));

    $number_of_divisors = 0;
    for ($i = 1; $i <= $x; $i++) {
        if ($x % $i == 0) {
            $number_of_divisors++;
        }
    }
    echo $number_of_divisors;
?>
```

This script, for input value 20, performs 20 iterations. However, wouldn't it be even better if it could perform almost the half of the iterations and achieve the same result? Of course it would! So, let's make it more efficient!

As you probably know, for any integer given (in variable $\$x$)

- the value 1 is always a divisor.
- the integer given is always a divisor of itself.
- there are no divisors after the middle of the range 1 to $\$x$.

Accordingly, for any integer there are certainly 2 divisors, the value 1 and the given integer itself. Therefore, the script must check for other possible divisors starting from the value 2 until the middle of the range 1 to $\$x$. The improved PHP script is as follows.

file_31.4-4b.php

```
<?php
    $x = trim(fgets(STDIN));

    $number_of_divisors = 2;
    for ($i = 2; $i <= intval($x / 2); $i++) {
        if ($x % $i == 0) {
            $number_of_divisors++;
        }
    }
    echo $number_of_divisors;
?>
```

This PHP script performs less than half of the iterations that the previous script did! To be more specific, for the input value 20, this PHP script performs only $(20 - 2) / 2 = 9$ iterations!

Exercise 31.4-5 *Is the Number a Prime?*

Write a PHP script that prompts the user to enter an integer greater than 1 and then displays a message indicating if this number is a prime. A prime number is any integer greater than 1 that has no divisors other than 1 and itself. The numbers 7, 11, and 13 are all such numbers.

Solution

This exercise is based on the previous one. It is very simple! If the given integer has only two divisors, the number is a prime. The PHP script is as follows.

file_31.4-5a.php

```
<?php
echo "Enter an integer greater than 1: ";
$x = trim(fgets(STDIN));

$number_of_divisors = 2;
for ($i = 2; $i <= intval($x / 2); $i++) {
    if ($x % $i == 0) {
        $number_of_divisors++;
    }
}

if ($number_of_divisors == 2) {
    echo "Number ", $x, " is prime";
}

?>
```

Now let's make the script more efficient. The flow of execution can break out of the loop when a third divisor is found, because this means that the given integer is definitely not a prime. There are two approaches.

First Approach – Using the break statement

The first approach uses the `break` statement, as shown in the next PHP script.

file_31.4-5b.php

```
<?php
echo "Enter an integer greater than 1: ";
$x = trim(fgets(STDIN));

$number_of_divisors = 2;
for ($i = 2; $i <= intval($x / 2); $i++) {
    if ($x % $i == 0) {
        $number_of_divisors++;
    }
}
```

```
        break;
    }
}

if ($number_of_divisors == 2) {
    echo "Number ", $x, " is prime";
}

?>
```

Second Approach – Using a pre-test loop structure

You can convert the counted loop structure to a pre-test loop structure and then change the Boolean expression of the loop to evaluate both variables `$i` and `$number_of_divisors`. As long as variable `$number_of_divisors` is equal to 2, the loop should continue iterating. The PHP script is as follows,

file_31.4-5c.php

```
<?php
echo "Enter an integer greater than 1: ";
$x = trim(fgets(STDIN));

$number_of_divisors = 2;
$i = 2;
while ($i <= intval($x / 2) && $number_of_divisors == 2) {
    if ($x % $i == 0) {
        $number_of_divisors++;
    }
    $i++;
}

if ($number_of_divisors == 2) {
    echo "Number ", $x, " is prime";
}

?>
```

Exercise 31.4-6 Finding all Prime Numbers from 1 to N

Write a PHP script that prompts the user to enter a positive integer and then displays all prime numbers from 2 to that given integer. Moreover, the script should validate data input and display an error message when the user enters any values less than 2.

Solution

To help you, this exercise is solved using the “from inner to outer” method.

The inner code fragment that checks whether variable `$x` contains a prime number has already been discussed in the previous exercise, so you can use it as is. On the other hand, the outer PHP script follows and is shown in general form.

```
echo "Enter an integer greater than 1: ";
$N = trim(fgets(STDIN));
while ($N < 2) {
    echo "Wrong number. Please enter an integer greater than 1: ";
    $N = trim(fgets(STDIN));
}

for ($x = 1; $x <= $N; $x++) {
    //Here goes the code that checks whether variable $x
    //contains a prime number
}
```

Thus, combining both scripts, the final PHP script becomes

file_31.4-6.php

```
<?php
echo "Enter an integer greater than 1: ";
$N = trim(fgets(STDIN));
while ($N < 2) {
    echo "Wrong number. Please enter an integer greater than 1: ";
    $N = trim(fgets(STDIN));
}

for ($x = 1; $x <= $N; $x++) {
    $number_of_divisors = 2;
    $i = 2;
    while ($i <= intval($x / 2) && $number_of_divisors == 2) {
        if ($x % $i == 0) {
            $number_of_divisors++;
        }
        $i++;
    }
    if ($number_of_divisors == 2) {
        echo "Number ", $x, " is prime\n";
    }
}

?>
```

From previous exercise

Exercise 31.4-7 Heron's Square Root

Write a PHP script that prompts the user to enter a non-negative value and then calculates its square root using Heron's formula, as follows.

$$x_{n+1} = \frac{\left(x_n + \frac{y}{x_n}\right)}{2}$$

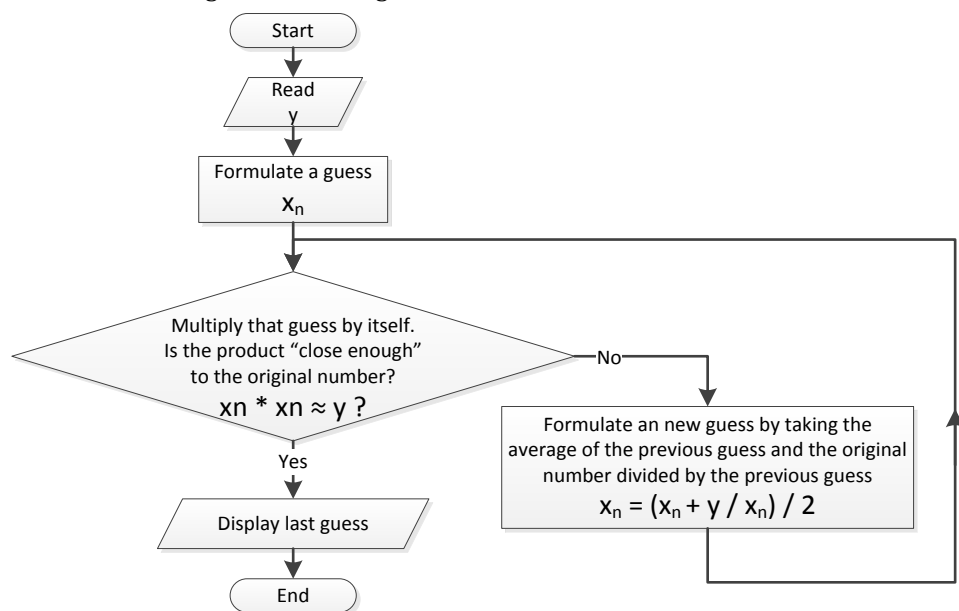
where

- y is the number for which you want to find the square root
- x_n is the n -th iteration value of the square root of y

Moreover, the script should validate data input and display an error message when the user enters any non-numeric or negative values.

Solution

It is almost certain that you are a little bit confused and you are scratching your head right now. Don't be scared by all this math stuff! You can try to understand Heron's formula through the following flowchart instead!



Still confused? Let's see an example. Let's find the square root of 25:

- Assume 8 as your first guess.
- The product of 8×8 is 64.
- Since 64 isn't "close enough" to 25, you can create a new guess by calculating the expression

$$\frac{\left(\text{guess} + \frac{\text{number}}{\text{guess}}\right)}{2} = \frac{\left(8 + \frac{25}{8}\right)}{2} \approx 5.56$$

- The product of 5.56×5.56 is about 30.91
- Since 30.91 isn't "close enough" to 25 you can create a new guess by calculating the expression

$$\frac{\left(\text{guess} + \frac{\text{number}}{\text{guess}}\right)}{2} = \frac{\left(5.56 + \frac{25}{5.56}\right)}{2} \approx 5.02$$

- The product of 5.02×5.02 is 25.2
- You can say that 25.2 is actually "close enough" to 25. Therefore, you can stop the whole process and say that the approximate square root of 25 is 5.02

Now, let's see the corresponding PHP script.

file_31.4-7.php

```
<?php
define("ACCURACY", 0.000000000001);

echo "Enter a non-negative number: ";
$y = trim(fgets(STDIN));
while (is_numeric($y) != true || $y < 0) {
    echo "Invalid value. Enter a non-negative number: ";
    $y = trim(fgets(STDIN));
}
$guess = rand(1, $y);    //Make a random first guess
                        //between 1 and given value

while (abs($guess * $guess - $y) > ACCURACY) { //is it "close enough"?
    $guess = ($guess + $y / $guess) / 2; //No, create a new "guess"!
}

echo $guess;
?>
```

Notice: Please note the way that "Is it close enough" is checked. When the absolute value of the difference $| \text{guess}^2 - \text{given\_value} |$ becomes less than 0.000000000001, the flow of execution exits the loop.

Exercise 31.4-8 Calculating π

Write a PHP script that calculates π using the Madhava–Leibniz^{2,3} series, which follows, with an accuracy of 0.00001.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

Solution

The Madhava–Leibniz series can be solved for π , and becomes

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \dots$$

The more fractions you have, the better the accuracy! Thus, to calculate this formula the script needs to perform many iterations so as to use as many fractions as possible. But, of course, it can't iterate forever! The loop should actually stop iterating when the current value of π and the one calculated in the previous iteration are "close enough," which means that the absolute value of their difference has become very small. The constant `ACCURACY` defines how small this difference should be. The PHP script is shown here.

file_31.4-8.php

```
<?php
define("ACCURACY", 0.00001);

$pi = 0;

$sign = 1;           //This is the sign of the first fraction
$denom = 1;          //This is the denominator of the first fraction
do {
    $pi_previous = $pi;
    $pi += $sign * 4 / $denom;
    $sign = -$sign;
    $denom += 2;
} while (abs($pi - $pi_previous) > ACCURACY); //is it "close enough"?
```

² Madhava of Sangamagrama (c. 1340–c. 1425), was an Indian mathematician and astronomer from the town of Sangamagrama (present day Irinjalakuda) of India. He founded the Kerala School of Astronomy and Mathematics and was the first to use infinite series approximations for various trigonometric functions. He is often referred to as the "father of mathematical analysis."

³ Gottfried Wilhelm von Leibniz (1646–1716) was a German mathematician and philosopher. He made important contributions to the fields of metaphysics, logic, and philosophy, as well as mathematics, physics, and history. In one of his works, *On the Art of Combination (Dissertatio de Arte Combinatoria)*, published in 1666, he formulated a model that is considered the theoretical ancestor of modern computers.


```
echo "Pi ~= ", $pi;
?>
```

Notice: Please note the way in which variable `$sign` toggles between the values `-1` and `+1` in each iteration.

If you reduce the value of the constant `ACCURACY`, π will be calculated more and more accurately. Depending on how fast your computer is, you can calculate the first six digits of π fairly quickly. However, the time it takes to calculate each succeeding digit of π goes up exponentially. To calculate 30 digits of π on a modern computer using this method could take years!

Exercise 31.4-9 *Approximating a Real with a Fraction*

Write a PHP script that prompts the user to enter a real between 1 and 100 and then tries to find the fraction $\frac{N}{M}$ that better approximates it, where N is an integer between 0 and 100 and M is an integer between 1 and 100. Moreover, the script should validate data input, allowing the user to enter only numeric values between 0 and 100. There is no need to display any error messages.

Solution

The solution is simple. All you need to do is iterate through all possible combinations of variables `$n` and `$m` and check which one better approximates the real given.

To iterate through all possible combinations of variables `$n` and `$m`, you can use a nested loop control structure, that is, two counted loop structures, one nested within the other, as follows.

```
for ($n = 0; $n <= 100; $n++) {
    for ($m = 1; $m <= 100; $m++)
        ...
}
```

Notice: The total number of iterations is $101 \times 100 = 10100$. Quite a big number but, for a modern computer, this is peanuts!

The following criteria

$$\text{minimum of } \left| \left(\frac{N}{M} - \text{real given} \right) \right|$$

can evaluate how “good” an approximation is.

Confused? Let’s try to approximate the value 0.333 with a fraction, iterating through all possible combinations of N and M .

- For $N = 1$, $M = 1$ the criteria equals to $\left| \frac{1}{1} - 0.333 \right| = 0.6670$

- For $N = 1$, $M = 2$ the criteria equals to $\left| \frac{1}{2} - 0.333 \right| = 0.1670$
- For $N = 1$, $M = 3$ the criteria equals to $\left| \frac{1}{3} - 0.333 \right| = 0.0003$
- For $N = 1$, $M = 4$ the criteria equals to $\left| \frac{1}{4} - 0.333 \right| = 0.0830$
- ...
- For $N = 100$, $M = 99$ the criteria equals to $\left| \frac{100}{99} - 0.333 \right| = 0.6771$
- For $N = 100$, $M = 100$ the criteria equals to $\left| \frac{100}{100} - 0.333 \right| = 0.6670$

As you can see, the value 0.0003 is the minimum value among all possible results. Thus, the combination $N = 1$ and $M = 3$ (which corresponds to the fraction $1/3$) is considered the best approximation for the value 0.333.

And now the PHP script:

file_31.4-9.php

```
<?php
do {
    echo "Enter a real between 0 and 100: ";
    $x = trim(fgets(STDIN));
} while (is_numeric($x) != true || $x < 0 || $x > 100);

$best_n = 1;
$best_m = 1;
for ($n = 0; $n <= 100; $n++) {
    for ($m = 1; $m <= 100; $m++) {
        if (abs($n / $m - $x) < abs($best_n / $best_m - $x)) {
            $best_n = $n;
            $best_m = $m;
        }
    }
}

echo "The corresponding fraction is: ", $best_n, "/", $best_m;
?>
```

31.5 Finding Minimum and Maximum Values with Loop Control Structures

In Chapter 23 you learned how to find the minimum and maximum values among four values using single-alternative decision structures. Now, the following code fragment does pretty much the same but uses only one variable `$w` to hold all of the user's given values.

```
$w = trim(fgets(STDIN));           //User enters 1st value
$max = $w;

$w = trim(fgets(STDIN));           //User enters 2nd value
if ($w > $max) {
    $max = $w;
}

$w = trim(fgets(STDIN));           //User enters 3rd value
if ($w > $max) {
    $max = $w;
}

$w = trim(fgets(STDIN));           //User enters 4th value
if ($w > $max) {
    $max = $w;
}
```

Except for the first pair of statements, all other blocks of statements are identical; therefore, only one of those can be enclosed within a loop control structure and be executed three times, as follows.

```
$w = trim(fgets(STDIN));           //User enters 1st value
$max = $w;

for ($i = 1; $i <= 3; $i++) {
    $w = trim(fgets(STDIN));       //User enters 2nd, 3rd and 4th value
    if ($w > $max) {
        $max = $w;
    }
}
```

Of course, if you want to allow the user to enter more values, you can simply increase the *final_value* of the counted loop structure.

Accordingly, if you want a script that finds and displays the heaviest person among 10 people, the solution is presented next.

file_31.5a.php

```
<?php
echo "Enter a weight: ";
$w = trim(fgets(STDIN));
$max = $w;

for ($i = 1; $i <= 9; $i++) {
```

```
echo "Enter a weight: ";
$w = trim(fgets(STDIN));
if ($w > $max) {
    $max = $w;
}

echo $max;

?>
```

Notice: Please note that the counted loop structure iterates one time less than the total number of values given.

Even though this PHP script operates perfectly well, let's do something different. Instead of prompting the user to enter one value before the loop and nine values inside the loop, let's prompt him or her to enter all values inside the loop.

The problem that arises here is that no matter what, before the loop starts iterating, an initial value must be assigned to variable `$max`. You can actually assign an “almost arbitrary” initial value but this value should be chosen carefully since a wrong one may lead to incorrect results.

In this exercise, all given values have to do with people's weight. Since there is no chance of finding any person with a negative weight (at least not on planet Earth), you can safely assign the initial value -1 to variable `$max`, as follows.

file_31.5b.php

```
<?php
$max = -1;

for ($i = 1; $i <= 10; $i++) {
    echo "Enter a weight: ";
    $w = trim(fgets(STDIN));
    if ($w > $max) {
        $max = $w;
    }
}

echo $max;

?>
```

As soon as the flow of execution enters the loop, the user enters the first value and the decision control structure validates to `true`. The value -1 in variable `$max` is overwritten by that first given value and after that, flow of execution proceeds normally.

Notice: Please note that this method cannot be used in every case. If an exercise needs to prompt the user to enter any number (not only a positive one) this method cannot be applied since the user could potentially enter only negative values. If this were to occur, the initial value -1 would never be overwritten by any of the given values. You can use this method to find the maximum value only when you know the lower limit of given values or to find the minimum value only when you know the upper limit of given values. For example, if the exercise needs to find the lightest person, you can assign the initial value +1500 to variable `$min`, since there is no human on Earth that can be so heavy! For the record, Jon Brower Minnoch was an American who, at his peak weight, was the heaviest human being ever recorded, weighing approximately 1.400 lb!!!!

Exercise 31.5-1 *Validating and Finding the Minimum and the Maximum Value*

Write a PHP script that prompts the user to enter the weight of 10 people and then finds the heaviest and the lightest one. Moreover, the script should validate data input and display an error message when the user enters any non-numeric, negative, zero value, or any values greater than 1500.

Solution

Using the previous exercise as a guide, you should now be able to do this with your eyes closed!

To validate data input, all you have to do is replace the following two lines of code of the previous exercise,

```
echo "Enter a weight: ";
$w = trim(fgets(STDIN));
```

with the following code fragment:

```
echo "Enter a weight: ";
$w = trim(fgets(STDIN));
while (is_numeric($w) != true || $w < 1 || $w > 1500) {
    echo "Invalid value! Please enter a weight between 1 and 1500!  \n";
    $w = trim(fgets(STDIN));
}
```

The final script is as follows.

`file_31.5-1.php`

```
<?php
$max = -1;
$min = 1500;

for ($i = 1; $i <= 10; $i++) {
    echo "Enter a weight: ";
```

```

$w = trim(fgets(STDIN));
while (is_numeric($w) != true || $w < 1 || $w > 1500) {
    echo "Invalid value! Please enter a weight between 1 and 1500! \n";
    $w = trim(fgets(STDIN));
}

if ($w > $max) {
    $max = $w;
}
if ($w < $min) {
    $min = $w;
}
}
echo $min, $max;
?>

```

Exercise 31.5-2 *Validating and Finding the Maximum Temperature*

Write a PHP script that prompts the user to repeatedly enter the names and the average temperatures of planets from space, until the word “STOP” (used as a name) is entered. In the end, the script should display the name of the hottest planet. Moreover, since -459.67° (on the Fahrenheit scale) is the lowest temperature possible (it is called absolute zero), the script should validate data input and display an error message when the user enters non-numeric values or values lower than absolute zero.

Solution

First, let’s write the PHP script without using data validation. According to the “Ultimate” rule, the pre-test loop structure should be as follows, given in general form:

```

echo "Enter the name of a planet: ";
$name = trim(fgets(STDIN));           //Initialization of $name
while (strtoupper($name) != "STOP") {
    A statement or block of statements

    echo "Enter the name of a planet: ";
    $name = trim(fgets(STDIN));       //Update/alteration of $name
}

```

Now, let’s add the rest of the statements, still without data input validation. Keep in mind that, since value -459.67° is the lower limit of the temperature scale, you can use a value lower than this as the initial value of variable `$max`.

```
<?php
$max = -460;
$m_name = "";

echo "Enter the name of a planet: ";
$name = trim(fgets(STDIN));
while (strtoupper($name) != "STOP") {
    echo "Enter its average temperature: ";
    $t = trim(fgets(STDIN));

    if ($t > $max) {
        $max = $t;
        $m_name = $name;
    }

    echo "Enter the name of a planet: ";
    $name = trim(fgets(STDIN));
}

if ($m_name != "") {
    echo "The hottest planet is: ", $m_name;
}
else {
    echo "Nothing Entered!";
}
?>
```

Notice: The `if ($m_name != "")` statement is required because there is a possibility that the user could enter the word “STOP” right from the beginning.

To validate the data input, all you have to do is replace the following two lines of code:

```
echo "Enter its average temperature: ";
$t = trim(fgets(STDIN));
```

with the following code fragment:

```
echo "Enter its average temperature: ";
$t = trim(fgets(STDIN));
while (is_numeric($t) != true || $t < -459.67) {
    echo "Invalid value!\n";
    echo "Enter an average temperature greater than -459.67 degrees!\n";
    $t = trim(fgets(STDIN));
}
```

```
}
```

The final script is as follows.

file_31.5-2.php

```
<?php
$max = -460;
$m_name = "";

echo "Enter the name of a planet/star: ";
$name = trim(fgets(STDIN));
while (strtoupper($name) != "STOP") {
    echo "Enter its average temperature: ";
    $t = trim(fgets(STDIN));
    while (is_numeric($t) != true || $t < -459.67) {
        echo "Invalid value!\n";
        echo "Enter an average temperature greater than -459.67 degrees!\n";
        $t = trim(fgets(STDIN));
    }

    if ($t > $max) {
        $max = $t;
        $m_name = $name;
    }

    echo "Enter the name of a planet/star: ";
    $name = trim(fgets(STDIN));
}

if ($m_name != "") {
    echo "The hottest planet/star is: ", $m_name;
}
else {
    echo "Nothing Entered!";
}

?>
```

Exercise 31.5-3 "Making the Grade"

In a classroom, there are 20 students. Write a PHP script that prompts the teacher to enter the grades (0 – 100) that students received in a math test and then displays the highest grade as well as the number of students that got an “A” (that is, 90 to 100).

Moreover, the script should validate data input. Given values should be within the range 0 to 100.

Solution

Let's first write the script without data validation. Since the number of students is known, you can use the counted loop structure. For an initial value of variable `$max`, you can use value -1 as there is no grade lower than 0.

```
<?php
$max = -1;
$count = 0;

for ($i = 1; $i <= 20; $i++) {
    echo "Enter a grade for student No ", $i, ": ";
    $grade = trim(fgets(STDIN));

    if ($grade > $max) {
        $max = $grade;
    }
    if ($grade >= 90) {
        $count++;
    }
}

echo $max, " ", $count;
?>
```

Now, you can deal with data validation. As the wording of the exercise implies, there is no need to display any error messages. So, all you need to do is replace the following two lines of code:

```
echo "Enter a grade for student No ", $i, ": ";
$grade = trim(fgets(STDIN));
```

with the following code fragment:

```
do {
    echo "Enter a grade for student No ", $i, ": ";
    $grade = trim(fgets(STDIN));
} while ($grade < 0 || $grade > 100);
```

and the final script becomes

file_31.5-3.php

```
<?php
$max = -1;
```

```

$count = 0;

for ($i = 1; $i <= 20; $i++) {
    do {
        echo "Enter a grade for student No ", $i, ": ";
        $grade = trim(fgets(STDIN));
    } while ($grade < 0 || $grade > 100);

    if ($grade > $max) {
        $max = $grade;
    }
    if ($grade >= 90) {
        $count++;
    }
}

echo $max, " ", $count;
?>

```

31.6 Exercises of a General Nature with Loop Control Structures

Exercise 31.6-1 *Fahrenheit to Kelvin, from 0 to 100*

Write a PHP script that displays all degrees Fahrenheit from 0 to 100 and their equivalent degrees Kelvin. Use an increment value of 0.5. It is given that

$$1.8 \cdot \text{Kelvin} = \text{Fahrenheit} + 459.67$$

Solution

The formula, solved for Kelvin becomes

$$\text{Kelvin} = \frac{\text{Fahrenheit} + 459.67}{1.8}$$

All you need here is a counted loop structure that increments the value of variable \$fahrenheit from 0 to 100 using an *offset* of 0.5. The solution is presented next.

file_31.6-1.php

```

<?php
for ($fahrenheit = 0; $fahrenheit <= 100; $fahrenheit += 0.5) {
    $kelvin = ($fahrenheit + 459.67) / 1.8;
    echo "Fahrenheit: ", $fahrenheit, " Kelvin: ", $kelvin, "\n";
}
?>

```

Exercise 31.6-2 *Wheat on a Chessboard*

On a chessboard you must place grains of wheat on each square, such that one grain is placed on the first square, two on the second, four on the third, and so on (doubling the number of grains on each subsequent square). How many grains of wheat will be on the chessboard in the end?

Solution

Assume a chessboard of only $2 \times 2 = 4$ squares and a variable `$grains` assigned the initial value 1 (this is the number of grains of the 1st square). A counted loop structure that iterates three times can double the value of variable `$grains` in each iteration, as shown in the next code fragment.

```
$grains = 1;
for ($i = 2; $i <= 4; $i++) {
    $grains = 2 * $grains;
}
```

The value of variable `$grains` at the end of each iteration is shown in the next table.

Iteration	Value of i	Value of grains
1 st	2	$2 \times 1 = 2$
2 nd	3	$2 \times 2 = 4$
3 rd	4	$2 \times 4 = 8$

At the end of the 3rd iteration, variable `$grains` contains the value 8. This value, however, is not the total number of grains on the chessboard but only the number of grains on the 4th square. If you need to find the total number of grains on the chessboard you can sum up the grains on all squares, that is, $1 + 2 + 4 + 8 = 15$.

In the real world a real chessboard contains $8 \times 8 = 64$ squares, thus you need to iterate for 64 times. The PHP script is as follows.

file_31.6-2.php

```
<?php
$grains = 1;
$sum = 1;
for ($i = 2; $i <= 64; $i++) {
    $grains = 2 * $grains;
    $sum = $sum + $grains;
}
echo $sum;
?>
```

In case you are wondering how big this number is, here is your answer: On the chessboard there will be 18,446,744,073,709,551,615 grains of wheat!

Exercise 31.6-3 *Just a Poll*

A public opinion polling company asks 1000 citizens if they eat breakfast in the morning. Write a PHP script that prompts the citizens to enter their sex (M for Male, F for Female) and their answer to the question (Y for Yes, N for No, S for Sometimes), and then calculates and displays the number of citizens that gave “Yes” as an answer, as well as the percentage of women among the citizens that gave “No” as an answer. Moreover, the script should validate data input and accept only values M or F for sex and Y, N, or S for answer.

Solution

The PHP script is as follows.

file_31.6-3.php

```
<?php
define("CITIZENS", 1000);

$total_yes = 0;
$female_no = 0;
for ($i = 1; $i <= CITIZENS; $i++) {
    do {
        echo "Enter Sex: ";
        $sex = trim(fgets(STDIN));
        $sex = strtolower($sex);
    } while ($sex != "m" && $sex != "f");

    do {
        echo "Do you eat breakfast in the morning? ";
        $answer = trim(fgets(STDIN));
        $answer = strtolower($answer);
    } while ($answer != "y" && $answer != "n" && $answer != "s");

    if ($answer == "y") {
        $total_yes++;
    }
    if ($sex == "f" && $answer == "n") {
        $female_no++;
    }
}
echo $total_yes, " ", $female_no * 100 / CITIZENS;

?>
```

Exercise 31.6-4 Is the Message a Palindrome?

A *palindrome* is a word or sentence that reads the same both backwards and forward. (You may recall from Chapter 23 that a number can also be a palindrome!) Write a PHP script that prompts the user to enter a word or sentence and then displays a message stating whether or not the given word or sentence is a palindrome. Following are some palindrome messages.

- Anna
- A nut for a jar of tuna.
- Dennis and Edna sinned.
- Murder for a jar of red rum.
- Borrow or rob?
- Are we not drawn onward, we few, drawn onward to new era?

Solution

There are some things that you should keep in mind before you start comparing the letters one by one and checking whether the first letter is the same as the last one, the second letter is the same as the last but one, and so on.

- In a given sentence or word, some letters may be in uppercase and some in lowercase. For example, in the sentence “A nut for a jar of tuna”, even though the first and last letters are the same, unfortunately they are not considered equal. Thus, the script must first change all the letters—for example, to lowercase—and then it can start comparing them.
- There are some characters such as spaces, periods, question marks, and commas that should be removed or else the script cannot actually compare the rest of the letters one by one. For example, in the sentence “Borrow or rob?” the PHP script may mistakenly assume that the sentence is not a palindrome because it will compare the first letter “B” with the last question mark “?”
- Assume that the sentence “Borrow or rob?”, after changing all letters to lowercase and after removing all unwanted spaces and the question mark, becomes “borroworrob”. These letters and their corresponding position in the string are as follows.

0	1	2	3	4	5	6	7	8	9	10
b	o	r	r	o	w	o	r	r	o	b

What you should keep in mind here is that the counted loop structure should iterate for only half of the letters. Can you figure out why?

The script starts the iterations and compares the letter at position 0 with the letter at position 10. Then it compares the letter at position 1 with the letter at position 9, and so on. The last iteration should be the one that compares the letters at positions 4 and 6. It is meaningless to continue checking thereafter, since all letters have already been compared.

One proposed solution is presented next. If you still have doubts about how this operates, you can use NetBeans IDE to execute it step by step and observe the values of the variables in each step.

file_31.6-4a.php

```
<?php
echo "Enter a message: ";
$message = trim(fgets(STDIN));

//Convert all letters to lowercase
$message = strtolower($message);

//This is the last position of $message
$last_pos = strlen($message) - 1;

//Create a new string which contains letters except
//spaces, commas, periods or questionmarks
$message_clean = "";
for ($i = 0; $i <= $last_pos; $i++) {
    if ($message[$i] != " " && $message[$i] != "," &&
        $message[$i] != "." && $message[$i] != "?") {

        $message_clean .= $message[$i];
    }
}

//This is the middle position of $message_clean
$middle_pos = intval((strlen($message_clean) - 1) / 2);

//This is the last position of $message_clean
$j = strlen($message_clean) - 1;
```

This code fragment can be further improved as you can see next.

```
//In the beginning, assume that sentence is palindrome
$palindrome = true;

//This counted loop structure compares
//letters one by one.
for ($i = 0; $i <= $middle_pos; $i++) {
```

```

    //If at least one pair of letters fails to validate
    //set variable $palindrome to false
    if ($message_clean[$i] != $message_clean[$j]) {
        $palindrome = false;
    }
    $j--;
}

//If variable $palindrome is still true
if ($palindrome == true) {
    echo "The message is palindrome";
}
?>

```

This PHP script works fine, but assume that the user enters a very large sentence that is not a palindrome. Let's say, for example, that the second letter is not the same as the last but one. Unfortunately, in this script the counted loop structure continues to iterate until the middle of the sentence despite the fact that variable `$palindrome` has been set to `false`, even from the second iteration. So, let's try to make this script even better.

What you need here is a way to break out of the loop when something goes wrong. There are two approaches, actually.

First Approach – Using the `break` statement

The first approach uses the `break` statement. In the initial PHP script, the statements marked with a dashed rectangle can be replaced with the following code fragment.

```

$palindrome = true;
for ($i = 0; $i <= $middle_pos; $i++) {
    if ($message_clean[$i] != $message_clean[$j]) {
        $palindrome = false;
        break;
    }
    $j--;
}

```

and the final PHP script becomes

file_31.6-4b.php

```

<?php
    echo "Enter a message: ";
    $message = trim(fgets(STDIN));

    //Convert all letters to lowercase
    $message = strtolower($message);

```

```
//This is the last position of $message
$last_pos = strlen($message) - 1;

//Create a new string which contains letters except
//spaces, commas, periods or questionmarks
$message_clean = "";
for ($i = 0; $i <= $last_pos; $i++) {
    if ($message[$i] != " " && $message[$i] != "," &&
        $message[$i] != "." && $message[$i] != "?") {

        $message_clean .= $message[$i];
    }
}

//This is the middle position of $message_clean
$middle_pos = intval((strlen($message_clean) - 1) / 2);

//This is the last position of $message_clean
$j = strlen($message_clean) - 1;

//In the beginning, assume that sentence is palindrome
$palindrome = true;

//This counted loop structure compares
//letters one by one.
for ($i = 0; $i <= $middle_pos; $i++) {
    //If at least one pair of letters fails to validate
    //set variable $palindrome to false
    if ($message_clean[$i] != $message_clean[$j]) {
        $palindrome = false;
        break;
    }
    $j--;
}

//If variable $palindrome is still true
if ($palindrome == true) {
    echo "The message is palindrome";
}
```



```
}
?>
```

Second Approach – Using a flag

To break out of a loop using a flag, always follow the same methodology: The counted loop structure must be converted to a pre-test loop structure and the Boolean expression of the loop must be modified to evaluate both variables `$i` and `$palindrome`. As long as variable `$palindrome` is `true`, the loop should continue iterating. In the initial PHP script, the statements marked with a dashed rectangle can be replaced with the following code fragment.

```
$palindrome = true;
$i = 0;
while ($i <= $middle_pos && $palindrome == true) {
    if ($message_clean[$i] != $message_clean[$j]) {
        $palindrome = false;
    }
    $j--;
    $i++;
}
```

and the final PHP script becomes

file_31.6-4c.php

```
<?php
echo "Enter a message: ";
$message = trim(fgets(STDIN));

//Convert all letters to lowercase
$message = strtolower($message);

//This is the last position of $message
$last_pos = strlen($message) - 1;

//Create a new string which contains letters except
//spaces, commas, periods or questionmarks
$message_clean = "";
for ($i = 0; $i <= $last_pos; $i++) {
    if ($message[$i] != " " && $message[$i] != "," &&
        $message[$i] != "." && $message[$i] != "?") {

        $message_clean .= $message[$i];
    }
}
```

```
}

//This is the middle position of $message_clean
$middle_pos = intval((strlen($message_clean) - 1) / 2);

//This is the last position of $message_clean
$j = strlen($message_clean) - 1;

//In the beginning, assume that sentence is palindrome
$palindrome = true;

//This counted loop structure compares
//letters one by one.
$i = 0;
while ($i <= $middle_pos && $palindrome == true) {
    if ($message_clean[$i] != $message_clean[$j]) {
        $palindrome = false;
    }
    $j--;
    $i++;
}

//If variable $palindrome is still true
if ($palindrome == true) {
    echo "The message is palindrome";
}

?>
```

Remember! Picture the variable `$palindrome` as a flag. Initially, the flag is “raised” (`$palindrome = true`). The flow of execution enters the loop and the loop continues iterating as long as the flag is up (while ... `$palindrome == true`). When something (usually a condition) lowers the flag (assigns value `false` to variable `$palindrome`), the flow of execution exits the loop.

31.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Data validation is the process of restricting data input, forcing the user to enter only valid values.
2. You can use a counted loop structure to validate data input.
3. You can use any loop control structure to validate data input.

4. To force a user to enter only positive numbers, without displaying any error messages, you can use the following code fragment.

```
do {  
    echo "Enter a positive number: ";  
    $x = trim(fgets(STDIN));  
} while (is_numeric($x) != true || $x <= 0);
```

5. To force a user to enter numbers between 1 and 10, you can use the following code fragment.

```
echo "Enter a number between 1 and 10: ";  
$x = trim(fgets(STDIN));  
while (is_numeric($x) == true && $x >= 1 && $x <= 10) {  
    echo "Wrong number \n";  
    echo "Enter a number between 1 and 10: ";  
    $x = trim(fgets(STDIN));  
}
```

6. When a PHP script is used to find the maximum value among 10 given values, the loop control structure that is used must always iterate 9 times.
7. In order to find the lowest number among 10 numbers given, you can use the following code fragment.

```
$min = 0;  
for ($i = 1; $i <= 10; $i++) {  
    $w = trim(fgets(STDIN));  
    if ($w < $min)  
        $min = $w;  
}
```

8. In order to find the highest number among 10 positive numbers given, you can use the following code fragment.

```
$max = 0;  
for ($i = 1; $i <= 10; $i++) {  
    $w = trim(fgets(STDIN));  
    if ($w > $max) {  
        $max = $w;  
    }  
}
```

31.8 Review Exercises

Complete the following exercises.

1. Write a PHP script that calculates and displays the sum of the following:

$$S = 1 + 3 + 5 + \dots + 99$$

2. Write a PHP script that lets the user enter an integer N and then calculates and displays the product of the following:

$$P = 2^1 \times 4^3 \times 6^5 \times \dots \times 2N^{(N-1)}$$

3. Write a PHP script that calculates and displays the sum of the following:

$$S = 1 + 2 + 4 + 7 + 11 + 16 + 22 + 29 + 37 + \dots + 191$$

4. Design a flowchart and write the corresponding PHP script that lets a teacher enter the total number of students as well as their grades and then calculates and displays the average value of those who got an “A”, that is 90 to 100. Add all necessary checks to make the script satisfy the property of definiteness.
5. Design a flowchart and write the corresponding PHP script that prompts the user to repeatedly enter numeric, non-negative values until their average value exceeds 3000. At the end, the script should display the total number of zeros entered.
6. Write a PHP script that prompts the user to enter an integer between 1 and 20 and then displays all four-digit integers for which the sum of their digits is less than the integer given. For example, if the user enters 15, the value 9301 is such a number, since

$$9 + 3 + 0 + 1 < 15$$

7. Write a PHP script that displays all four-digit integers that satisfy all of the following conditions:

- the number’s first digit is greater than its second digit
- the number’s second digit is equal to its third digit
- the number’s third digit is smaller than its fourth digit

For example, the values 7559, 3110, and 9889 are such numbers.

8. A student wrote the following code fragment which is supposed to validate data input, forcing the user to enter only values 0 and 1. Identify any error(s) in the code fragment.

```
while ((is_numeric($x) != true) && ($x != 1) && ($x != 0)) {
    echo "Error!";
    $x = trim(fgets(STDIN));
}
```

9. Write the code fragment that validates data input, forcing the user to enter a valid sex (M for Male, F for Female). Moreover, it should validate correctly both for lowercase and uppercase letters.
10. Write a PHP script that prompts the user to enter a non-negative number and then calculates its square root. Moreover, the script should validate data

input and display an error message when the user enters any non-numeric or negative values. Additionally, the user has a maximum number of two retries. If the user enters more than three non-numeric or negative values, a message “Dude, you are dumb!” should be displayed and the script execution should end.

11. The area of a circle can be calculated using the following formula:

$$\text{Area} = \pi \cdot \text{Radius}^2$$

Write a PHP script that prompts the user to enter the length of the radius of a circle and then calculates and displays its area. The script should iterate as many times as the user wishes. At the end of each area calculation, the script should ask the user if he or she wishes to calculate the area of another circle. If the answer is “yes” the script should repeat; it should end otherwise. Make your script accept the answer in all possible forms such as “yes”, “YES”, “Yes”, or even “YeS”.

Moreover, the script should validate data input and display an error message when the user enters any non-numeric or non-positive value for Radius.

Hint: Use the `pi()` function to get the value of π .

12. Write a PHP script that displays all possible integer values of x and y within the range -100 to +100 that validate (when possible) the following formula:

$$5x + 3y^2 = 0$$

13. Write a PHP script that displays all possible integer values of x , y , and z within the range -10 to +10 that validate the following formula:

$$\frac{x + y}{2} + \frac{3z^2}{x + 3y + 45} = \frac{x}{3}$$

14. Write a PHP script that lets the user enter three positive integers and then finds their product using the Russian multiplication algorithm.
15. Write a PHP script that prompts the user to enter two positive integers and then displays all prime integers between them. Moreover, the script should validate data input and display an error message when the user enters a value less than +2.

Hint: To make your PHP script operate correctly, independent of which integer given is the lowest, you can swap their values (if necessary) so that they are always in the proper order.

16. Write a PHP script that prompts the user to enter two positive four-digit integers and then displays all integers between them that are palindromes. Moreover, the script should validate data input and display an error message when the user enters any non-numeric characters, or any numbers other than four-digit ones.

Hint: To make your PHP script operate correctly, independent of which integer given is the lowest, you can swap their values (if necessary) so that they are always in the proper order.

17. Write a PHP script that displays all possible RAM sizes between 1 byte and 1GByte, such as 1, 2, 4, 8, 16, 32, 64, 128, and so on.
Hint: 1GByte equals 2^{30} bytes, or 1073741824 bytes

18. Write a PHP script that displays the following sequence of numbers:

1, 11, 23, 37, 53, 71, 91, 113, 137, ... 401

19. Write a PHP script that displays the following sequence of numbers:

-1, 1, -2, 2, -3, 3, -4, 4, ... -100, 100

20. Write a PHP script that displays the following sequence of numbers:

1, 11, 111, 1111, 11111, ... 11111111

21. The Fibonacci⁴ sequence is a series of numbers in the following sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

By definition, the first two numbers are 1 and 1 and each subsequent number is the sum of the previous two.

Write a PHP script that lets the user enter a positive integer and then displays as many Fibonacci numbers as that given integer.

22. Write a PHP script that lets the user enter a positive integer and then displays all Fibonacci numbers that are less than that given integer.

23. Write a PHP script that prompts the user to enter a positive integer N and then finds and displays the value of y in the following formula:

$$y = \frac{2 + 4 + 6 + \dots + 2N}{1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot N}$$

Moreover, the script should validate data input and display an error message when the user enters any non-numeric value or a value less than 1.

24. Write a PHP script that prompts the user to enter an integer N and then finds and displays the value of y in the following formula

$$y = \frac{1 - 3 + 5 - 7 + \dots + (2N + 1)}{N}$$

Moreover, the script should validate data input and display an error message when the user enters any non-numeric value or a value less than 1.

⁴ Leonardo Pisano Bigollo (c. 1170–c. 1250), also known as Fibonacci, was an Italian mathematician. In his book *Liber Abaci* (published in 1202), Fibonacci used a special sequence of numbers to try to determine the growth of a rabbit population. Today, that sequence of numbers is known as the Fibonacci sequence. He was also one of the first people to introduce the Arabic numeral system to Europe; this is the numeral system we use today, based on ten digits with a decimal point and a symbol for zero. Before then, the Roman numeral system was being used, making numerical calculations difficult.

25. Write a PHP script that prompts the user to enter an integer N and then finds and displays the value of y in the following formula:

$$y = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{5} + \dots + \frac{1}{N}$$

Moreover, the script should validate data input and display an error message when the user enters any non-numeric value or a value less than 1.

26. Write a PHP script that prompts the user to enter an integer N and then finds and displays the value of y in the following formula:

$$y = \frac{1}{1^N} + \frac{1}{2^{N-1}} + \frac{1}{3^{N-2}} + \dots + \frac{1}{N^1}$$

Moreover, the script should validate data input and display an error message when the user enters any non-numeric value or a value less than 1.

27. In mathematics, the factorial of a non-negative integer N is the product of all positive integers less than or equal to N , and it is denoted by $N!$. The factorial of 0 is, by definition, equal to 1. In mathematics, you can write

$$N! = \begin{cases} 1 \cdot 2 \cdot 3 \cdot \dots \cdot N, & \text{for } N > 0 \\ 1, & \text{for } N = 0 \end{cases}$$

For example, the factorial of 5 is $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$.

Write a PHP script that prompts the user to enter a non-negative integer N and then calculates its factorial.

28. Write a PHP script that lets the user enter a value for x and then calculates and displays the exponential function e^x using the Taylor⁵ series, shown next, with an accuracy of 0.00001.

$$e^x = 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Hint: Keep in mind that $\frac{x^0}{0!} = 1$.

29. Write a PHP script that lets the user enter a value for x and then calculates and displays the sine of x using the Taylor series, shown next, with an accuracy of 0.00001.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Hint: Keep in mind that x is in radians and $\frac{x^1}{1!} = x$.

⁵ Brook Taylor (1685–1731) was an English mathematician who is best known for the Taylor series and his contributions to the theory of finite differences.

30. Write a PHP script that lets the user enter a value for x and then calculates and displays the cosine of x using the Taylor series, shown next, with an accuracy of 0.00001.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

Hint: Keep in mind that x is in radians and $\frac{x^0}{0!} = 1$.

31. Write a PHP script that prompts the user to enter the daily temperatures in August and then calculates and displays the average as well as the highest temperature.

Moreover, since -459.67° (on the Fahrenheit scale) is the lowest temperature possible (it is called absolute zero), the script should validate data input and display individual error messages when the user enters a non-numeric or a value lower than absolute zero.

32. A scientist needs a software application to record the level of the sea based on values recorded at specific times (HH:MM), in order to extract some useful information. Write a PHP script that lets the scientist enter the level of the sea, the hour, and the minutes, repeatedly until the value of 9999 (used as a sea level) is entered. Then, the script should display the highest and the lowest sea levels as well as the hour and the minutes at which these levels were recorded.
33. Suppose that the letter A corresponds to the number 1, the letter B corresponds to the number 2, and so on. Write a PHP script that prompts the user to enter two integers and then displays all alphabet letters that exist between them. For example, if the user enters 3 and 6, the script should display C, D, E, F. Moreover, the script should validate data input and display individual error messages when the user enters any non-numeric, negative, or any value greater than 26.

Hint: To make your PHP script operate correctly, independent of which integer given is the lowest, you can swap their values (if necessary) so that they are always in the proper order.

34. Write a PHP script that assigns a random secret integer between 1 and 100 to a variable and then prompts the user to guess the number. If the integer given is less than the secret one, a message *"Your guess is smaller than my secret number. Try again."* should be displayed. If the integer given is greater than the secret one, a message *"Your guess is bigger than my secret number. Try again."* should be displayed. This process should repeat until the user finally finds the secret number. Then, a message *"You found it!"* should be displayed, as well as the total number of the user's attempts.
35. Expand the previous exercise/game by making it operate for two players. The player that wins is the one that finds the random secret number in fewer attempts.
36. The size of a TV screen always refers to its diagonal measurement. For example, a 40-inch TV screen is 40 inches diagonally, from one corner on top

to the other corner on bottom. The old TV screens had a width-to-height aspect ratio of 4:3, which means that for every 3 inches in TV screen height, there were 4 inches in TV screen width. Today, most TV screens have a width-to-height aspect ratio of 16:9, which means that for every 9 inches in TV screen height there are 16 inches in TV screen width. Using these aspect ratios and the Pythagorean Theorem, you can easily determine that:

➤ **for all 4:3 TVs**

Width = Diagonal $\times$ 0.8

Height = Diagonal $\times$ 0.6

➤ **for all 16:9 TVs**

Width = Diagonal $\times$ 0.87

Height = Diagonal $\times$ 0.49

Write a PHP script that displays the following menu:

1. 4/3 TV Screen
2. 16/9 TV Screen
3. Exit

and prompts the user to enter a choice as well as the diagonal screen size in inches. Then, the PHP script should display the width and the height of the TV screen. This process will continue repeatedly, until the user selects choice 3 (Exit) from the menu.

37. Write a PHP script that prompts a teacher to enter the total number of students, their grades, and their sex (M for Male, F for Female), and then calculates and displays all of the following:
- a. the average value of those who got an "A" (90 - 100)
 - b. the average value of those who got a "B" (80 - 89)
 - c. the average value of boys who got an "A" (90 - 100)
 - d. the total number of girls that got less than "B"
 - e. the highest and lowest grade
 - f. the average grade of the whole class

Add all necessary checks to make the script satisfy the property of definiteness. Moreover, the script should validate data input and display an error message when the teacher enters any of the following:

- non-numeric or non-positive values for total number of students
- values other than M or F for sex
- non-numeric, negatives, or values greater than 100 for student grades

38. Write a PHP script that calculates and displays the discount that a customer receives based on the amount of his or her order, according to the following table.

Amount	Discount
$\$0 < \text{amount} < \$20$	0%
$\$20 \leq \text{amount} < \$50$	3%
$\$50 \leq \text{amount} < \$100$	5%
$\$100 \leq \text{amount}$	10%

At the end of each discount calculation, the script should ask the user if he or she wishes to calculate the discount of another amount. If the answer is “yes,” the script should repeat; it should end otherwise. Make your script accept the answer in all possible forms such as “yes”, “YES”, “Yes”, or even “YeS”.

Moreover, the script should validate data input and display an error message when the user enters any non-numeric or non-positive value for amount.

39. The LAV Electricity Company charges subscribers for their electricity consumption according to the following table (monthly rates for domestic accounts).

Kilowatt-hours (kWh)	USD per kWh
$0 \leq \text{kWh} \leq 400$	\$0.11
$401 \leq \text{kWh} \leq 1500$	\$0.22
$1501 \leq \text{kWh} \leq 3500$	\$0.25
$3501 \leq \text{kWh}$	\$0.50

Write a PHP script that prompts the user to enter the total number of kWh consumed by a subscriber and then calculates and displays the total amount to pay. This process continues until the value -1 is entered.

Moreover, the script should validate data input and display an error message when the user enters any non-numeric or negative value for kWh. An exception for the value -1 should be made.

Transmission services and distribution charges, as well as federal, state, and local taxes, add a total of 25% to each bill.

Please note that the rates are progressive.

Review Questions in “Loop Control Structures”

Answer the following questions.

1. What is a loop control structure?
2. In a flowchart, how can you distinguish a decision control structure from a loop control structure?
3. Design the flowchart and write the PHP statement (in general form) of a pre-test loop structure. Explain how this loop control structure operates.
4. Why is a pre-test loop structure named this way, and what is the fewest number of iterations it may perform?
5. If the statement or block of statements of a pre-test loop structure is executed N times, how many times is the Boolean expression of the structure evaluated?
6. Design the flowchart and write the corresponding PHP statement (in general form) of a post-test loop structure. Explain how this loop control structure operates.
7. Why is a post-test loop structure named this way, and what is the fewest number of iterations it may perform?
8. If the statement or block of statements of a post-test loop structure is executed N times, how many times is the Boolean expression of the structure evaluated?
9. Design the flowchart and write the corresponding PHP statement (in general form) of a counted loop structure. Explain how this loop control structure operates.
10. State the rules that apply to counted loop structures.
11. What are nested loops?
12. Write an example of a script with nested loop control structures and explain how they are executed.
13. State the rules that apply to nested loops.
14. Design a diagram that could help someone decide on the best loop control structure to choose.
15. Describe the “Ultimate” rule and give an example, in general form, using both a pre-test and a post-test loop structure.
16. Suppose that a PHP script uses a loop control structure to search for a given word in an electronic dictionary. Why is it critical to break out of the loop when the given word is found?
17. Why is it critical to clean out your loops?
18. What is an infinite loop?
19. Describe, using PHP statements in general form, how to do a conversion from a counted loop structure to a pre-test loop structure.

20. Describe, using PHP statements in general form, how to do a conversion from a pre-test loop structure to a counted loop structure.
21. Describe, using PHP statements in general form, how to do a conversion from a post-test loop structure to a pre-test loop structure.
22. Describe, using PHP statements in general form, how to do a conversion from a pre-test loop structure to a post-test loop structure.
23. Describe, using PHP statements in general form, how to do a conversion from a counted loop structure to a post-test loop structure.
24. Describe, using PHP statements in general form, how to do a conversion from a post-test loop structure to a counted loop structure.

Section 6

Arrays

Chapter 32

Introduction to Arrays

32.1 Introduction

Variables are a good way to store values in memory but they have one limitation—they can hold only one value at a time. There are many cases, however, where a script needs to process long lists of data. For example, consider the following exercise.

Write a PHP script that lets the user enter the grades of 20 students (on a scale of 0–100) and then displays

- i. the highest grade; and*
- ii. the total number of students that have a grade equal to the highest grade.*

Finding the highest grade is easy and has already been discussed in Chapter 31. The code fragment is as follows.

```
$max_grade = -1;
for ($i = 1; $i <= 20; $i++) {
    $grade = trim(fgets(STDIN));
    if ($grade > $max_grade) {
        $max_grade = $grade;
    }
}

echo $max_grade;
```

However, it becomes difficult to determine how many students have a grade equal to that highest grade because:

- while the loop iterates, the highest grade has not yet been found, and
- when the loop finally finishes iterating and the highest grade has been found, variable `$grade` contains only that last value given. Unfortunately, all previous 19 grades have been lost, and it just doesn't seem right to have to prompt the user to enter all the grades again!

The way this script is currently written, it is impossible to compare the value of variable `$max_grade` to all 20 values given, and to count those that are equal to that value.

What you need here is a way to keep each and every student grade in main memory (RAM) as the user inputs them so that the script can use them later; however, this is not possible for this PHP script because of the way it is currently written. One possible solution would be to use 20 individual variables, as shown here.

```
<?php
```

```
$a1 = trim(fgets(STDIN));
$a2 = trim(fgets(STDIN));
$a3 = trim(fgets(STDIN));
...
$a20 = trim(fgets(STDIN));
```

Input 20 values into 20 individual variables

```
$max = $a1;  
if ($a2 > $max) {  
    $max = $a2;  
}  
if ($a3 > $max) {  
    $max = $a3;  
}  
...  
if ($a20 > $max) {  
    $max = $a20;  
}
```

Find maximum value

```
$count = 0;  
if ($a1 == $max) {  
    $count++;  
}  
if ($a2 == $max) {  
    $count++;  
}  
if ($a3 == $max) {  
    $count++;  
}  
...  
if ($a20 == $max) {  
    $count++;  
}
```

Count all those that are equal to \$max

```
echo $count;
```

```
?>
```

This script appears to work fine, but what if 2000 instead of 20 grades have to be entered? Think about it—do you have the patience to write an analogous PHP script for 2000 grades? Of course not! Fortunately for you, there are arrays!

32.2 What is an Array?

An array is a special type of variable that can hold multiple values at the same time under one common name. An array can be thought of as a collection of positions, each containing one value, organized in rows, columns, or a matrix. Some examples of arrays are as follows.

64	71	74	63	61	64	57

57	58	65	71	75	68	56
64	71	74	63	61	64	57
66	61	71	62	74	64	78
73	60	67	54	59	62	64

Maria
George
John
Tom
Andrew
Helen

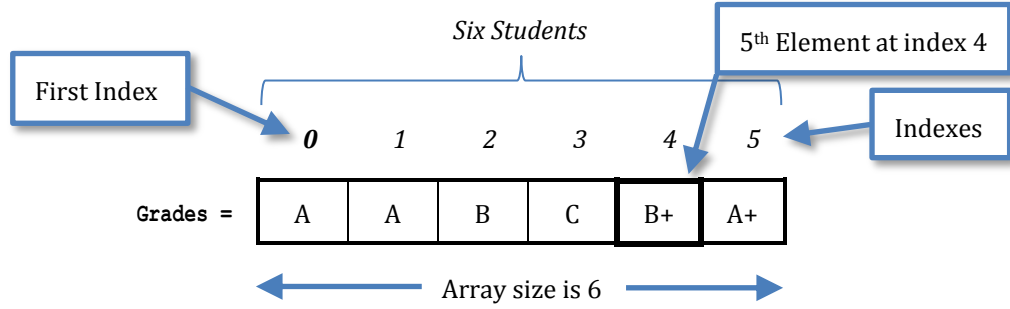
Each item in an array is called an element. In memory, array elements are located in consecutive memory locations. Each element in an array is assigned a unique number known as its index position, or simply an index.

Notice: Arrays in computer science resemble the matrices used in mathematics. A mathematical matrix is a collection of numbers or other mathematical objects, arranged in rows and columns.

There are one-dimensional and multidimensional arrays. A multidimensional array can be two-dimensional, three-dimensional, four-dimensional, and so on.

One-Dimensional Arrays

The following example presents a one-dimensional array that holds the grades of six students. The name of the array is `Grades`. For your convenience, the corresponding index is written above each element. By default, in PHP, index numbering always starts at zero.



Remember! Since index numbering starts at zero, the index of the last element of an array is 1 less than the total number of elements in the array.

You can think of an array as if it were six individual variables—`Grades0`, `Grades1`, `Grades2`, ... `Grades5`—with each variable holding the grade of one student. The advantage of the array, however, is that it can hold multiple values under one common name.

Notice: In PHP, this type of array is often called an indexed array. PHP also supports associative arrays but they are beyond the scope of this book and so are purposely not mentioned. If you need more information, however, you can visit

<http://www.php.net/manual/en/language.types.array.php>

Two-Dimensional Arrays

In general, multidimensional arrays are useful for working with multiple sets of data. For example, suppose you want to hold the daily high temperatures for California for the first two weeks of April. One approach would be to use two one-dimensional arrays, one for each week. Furthermore, each array would have seven elements, one for each day of the week, as follows.

	Days						
	0	1	2	3	4	5	6
Temperatures_week1 =	57	58	65	71	75	68	56
Temperatures_week2 =	64	71	74	63	61	64	57

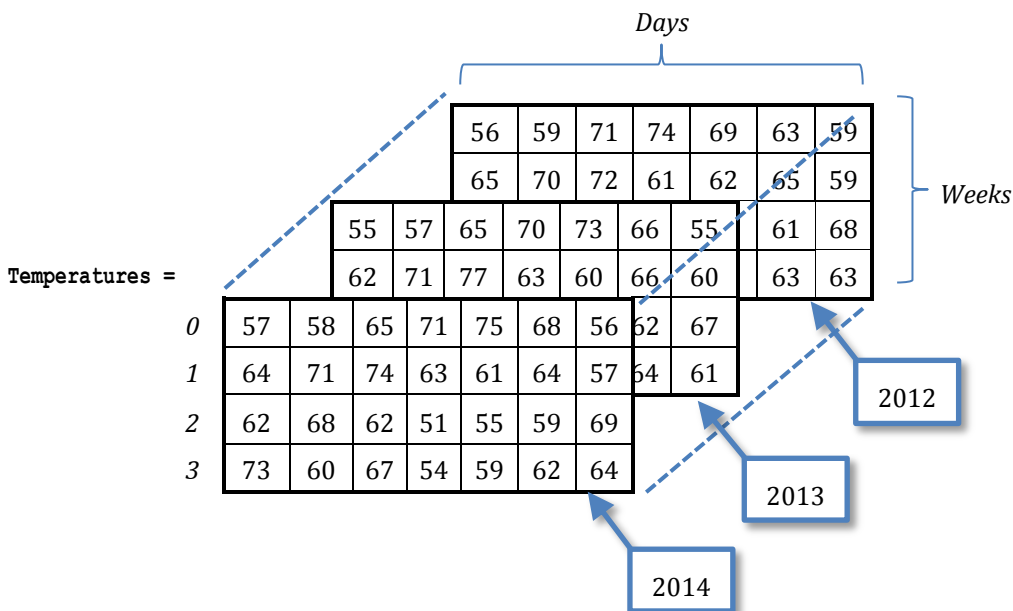
However, this approach is a bit awkward because you would have to process each array separately. A better approach would be to use a two-dimensional array with two rows (one for each week) and seven columns (one for each day of the week), as follows.

		Days						
		0	1	2	3	4	5	6
Temperatures =	0	57	58	65	71	75	68	56
	1	64	71	74	63	61	64	57

Weeks

Three-Dimensional Arrays

The next example shows a three-dimensional array that holds the daily high temperatures for California for the four weeks of April for the years 2012, 2013, and 2014.



Notice: Please note that four-dimensional, five-dimensional, or even one-hundred-dimensional arrays can exist. However, experience shows that the maximum array dimension that you will need in your life as a programmer is probably two or three.

Exercise 32.2-1 Designing an Array

Design an array that can hold the ages of 10 people, and then add some typical values to the array.

Solution

This is an easy one. All you have to do is design an array with 10 positions (indexes 0 to 9). It can be an array with either one row or one column, as follows.

People									
	0	1	2	3	4	5	6	7	8
Ages =	17	25	45	12	5	60	33	23	21

Ages =	17	0	} People
	25	1	
	45	2	
	12	3	
	5	4	
	60	5	
	33	6	
	23	7	
	21	8	
	49	9	

Keep in mind, however, that there are no arrays with one row or one column in PHP. These concepts may exist in mathematical matrices (or in your imagination!) but not in PHP. The arrays in PHP are one-dimensional—end of story! If you want to visualize them having one row or one column, that is up to you.

Exercise 32.2-2 *Designing Arrays*

Design the necessary arrays to hold the names and the ages of eight people, and then add some typical values to the arrays.

Solution

This exercise can be implemented with two arrays. Let’s design them with one column each.

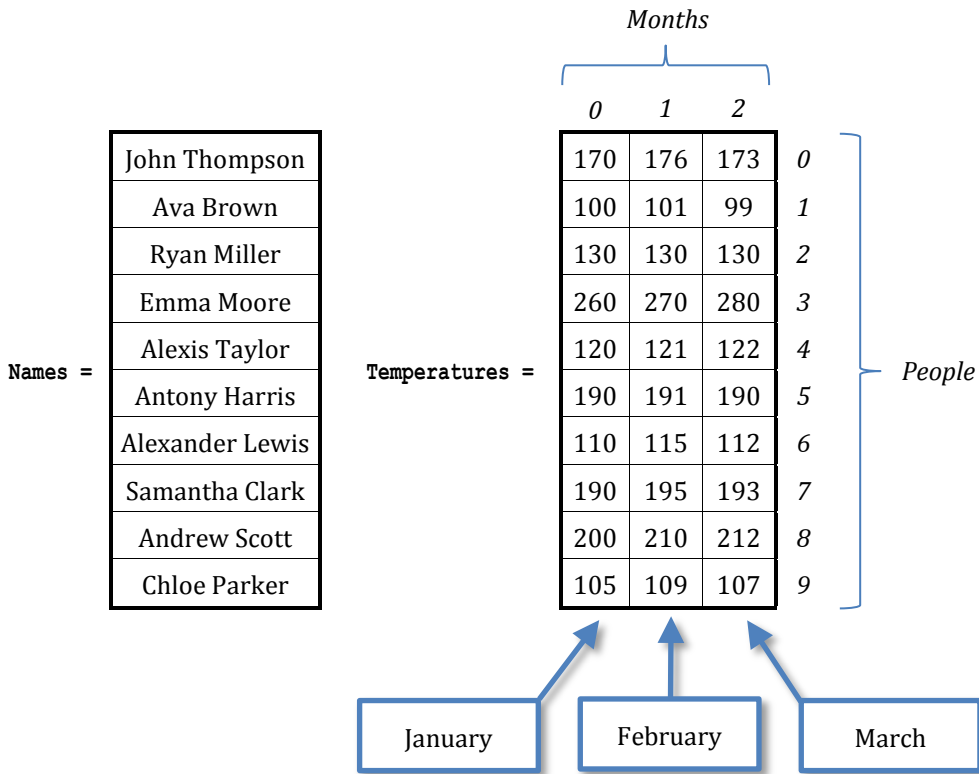
Names =	John Thompson	Ages =	17	0	} People
	Ava Brown		25	1	
	Ryan Miller		45	2	
	Antony Harris		60	3	
	Alexander Lewis		33	4	
	Samantha Clark		23	5	
	Andrew Scott		21	6	
	Chloe Parker		49	7	

Exercise 32.2-3 *Designing Arrays*

Design the necessary arrays to hold the names of ten people as well as the average weight (in pounds) of each person for January, February, and March. Then add some typical values to the arrays.

Solution

In this exercise, you need a one-dimensional array for names, and a two-dimensional array for people's weights.



32.3 **Review Questions: True/False**

Choose **true** or **false** for each of the following statements.

1. Arrays are special forms of variables that can hold multiple values at the same time.
2. In memory, array elements are located in consecutive memory locations.
3. There can be only one-dimensional and two-dimensional arrays.
4. There cannot be four-dimensional arrays.

5. An array is called “multidimensional” because it can hold values of different types.
6. Each element of an array has a unique index.
7. By default, index numbering always starts at zero.
8. The index of the last element of an array is equal to the total number of elements in the array.
9. A two-hundred-dimensional array can exist.

32.4 Review Exercises

Complete the following exercises.

1. Design an array to hold the weights of five people, and then add some typical values to the array.
2. Design the necessary arrays to hold the names and the weights (in pounds) of seven people, and then add some typical values to the arrays.
3. Design the necessary arrays to hold the names of five lakes as well as the average area (in square miles) of each lake in June, July, and August. Then add some typical values to the arrays.
4. Design an array to hold the three dimensions (width, height, and depth in inches) of 10 boxes. Then add some typical values to the array.
5. Design the necessary arrays to hold the names of eight lakes as well as the average area (in square miles) and maximum depth (in feet) of each lake. Then add some typical values to the arrays.
6. Design the necessary arrays to hold the names of four lakes as well as their average areas (in square miles) for the first week of June, the first week of July, and the first week of August.

Chapter 33

One-Dimensional Arrays

33.1 Creating One-Dimensional Arrays in PHP

There are several approaches that can be used to create and add values (elements) to an array in PHP. Which one to use is up to you, depending on the given problem.

Let's try to create the following array, `$grades`, using all approaches.

	0	1	2	3
<code>\$grades =</code>	B+	A+	A	C-

First Approach

You can create an array in PHP using the following statement given in general form:

```
$array_name = array();
```

and then you can add a value (an element) to it using the following statement, given in general form:

```
$array_name[index] = value;
```

where *index* is the index position of the element in the array.

The next code fragment creates the array `$grades` and adds some values (elements) to it.

```
$grades = array();  
$grades[0] = "B+";  
$grades[1] = "A+";  
$grades[2] = "A";  
$grades[3] = "C-";
```

Notice: In Chapter 5 you learned about the rules that must be followed when assigning names to PHP variables. Assigning names to PHP arrays follows exactly the same rules!

Of course, instead of using constant values for *index*, you can also use variables or expressions, as follows.

```
$grades = array();  
$k = 0;  
$grades[$k] = "B+";  
$grades[$k + 1] = "A+";  
$grades[$k + 2] = "A";  
$grades[$k + 3] = "C-";
```

Second Approach

The same result can be achieved without using an *index*, as follows.

```
$grades = array();  
$grades[] = "B+";  
$grades[] = "A+";  
$grades[] = "A";
```



```
$grades[] = "C-";
```

Notice: Please note that in this approach, index numbering starts at zero by default. This means that grade B+ is added to index position 0.

Third Approach

Another way to create an array and directly add values to it is shown in the next PHP statement, given in general form.

```
$array_name = array(value0, value1, value2, ..., valueM);
```

Using this approach, the array \$grades can be created using the following statement:

```
$grades = array("B+", "A+", "A", "C-");
```

Notice: Please note that in this approach as well, index numbering starts at zero by default.

Fourth Approach

Last but not least, if you wish to create an array and directly add some elements to certain positions you can use the next PHP statement, given in general form.

```
$array_name = array(index0 => value0, index1 => value1, ..., indexM => valueM);
```

Thus, the array \$grades can be created using the following statement:

```
$grades = array(0 => "B+", 1 => "A+", 2 => "A", 3 => "C-");
```

In this approach, elements can also be in random order.

```
$grades = array(3 => "C-", 1 => "A+", 0 => "B+", 2 => "A");
```

As previously shown, instead of using constant values for *index*, you can also use variables or expressions as shown next.

```
$k = 0;
$grades = array($k + 3 => "C-", $k + 1 => "A+", $k => "B+", $k + 2 => "A");
```

33.2 How to Get Values from One-Dimensional Arrays

Getting values from arrays is just a matter of pointing to a specific position. Each element of a one-dimensional array can be uniquely identified using an index. The following code fragment

```
$grades = array("B+", "A+", "A", "C-");
echo $grades[1];
```

displays A+ on the screen.

Of course, instead of using constant values for *index*, you can also use variables or expressions. The following example

```
$grades = array("B+", "A+", "A", "C-");
$k = 3;
echo $grades[$k], " and ", $grades[$k - 1];
```

displays C- and A on the screen.

Exercise 33.2-1 *Creating the Trace Table*

Create the trace table for the next code fragment. At the end, the array should contain four elements.

```
$a = array();
$a[3] = 9;
$x = 0;
$a[$x] = $a[3] + 4;
$a[$x + 1] = $a[$x] * 3;
$x++;
$a[$x + 2] = $a[$x - 1];
$a[2] = $a[1] + 5;
$a[3] = $a[3] + 1;
```

Solution

Don't forget that you can manipulate each element of an array as if it were a variable. Thus, when you create a trace table for a PHP script that uses an array, you can have one column for each element as follows.

Step	Statement	Notes	\$x	\$a[0]	\$a[1]	\$a[2]	\$a[3]
1	\$a = array()	This creates array \$a with no values in it	?	?	?	?	?
2	\$a[3] = 9		?	?	?	?	9
3	\$x = 0		0	?	?	?	9
4	\$a[\$x] = \$a[3] + 4		0	13	?	?	9
5	\$a[\$x + 1] = \$a[\$x] * 3		0	13	39	?	9
6	\$x++		1	13	39	?	9
7	\$a[\$x + 2] = \$a[\$x - 1]		1	13	39	?	13
8	\$a[2] = \$a[1] + 5		1	13	39	44	13
9	\$a[3] = \$a[3] + 1		1	13	39	44	14

Exercise 33.2-2 *Using a Non-Existing Index*

Which properties of an algorithm are not satisfied by the following PHP script?

```
<?php
    $grades = array("B+", "A+", "A", "C-");
    echo $grades[100];
?>
```

Solution

Two properties are not satisfied by this algorithm. The first one is obvious: there is no data input. The second one is the property of definiteness. You should never refer to a non-existing element of an array and, in this exercise, there is no element at index position 100.

33.3 How to Add Values Entered by the User to a One-Dimensional Array

There is nothing new here. Instead of reading a value from the keyboard and assigning that value to a variable, you can directly assign that value to a specific position of an array. The next code fragment lets the user enter three values, and then adds these values to positions 0, 1, and 2 of the array `$test`.

```
$test = array();  
$test[0] = trim(fgets(STDIN));  
$test[1] = trim(fgets(STDIN));  
$test[2] = trim(fgets(STDIN));
```

33.4 How to Iterate Through a One-Dimensional Array

Now comes the interesting part. A script can iterate through the elements of an array using a loop control structure (usually a counted loop structure) in which the *counter* variable is used as the *index* to the array. Following is a code fragment, written in general form

```
for ($i = 0; $i <= size_of_the_array - 1; $i++) {  
    process array_name[$i];  
}
```

in which *size_of_the_array* is the number of elements the array contains, and *process* is any PHP statement or block of statements that processes one element of the array at each iteration.

Notice: The name of the variable `$i` is not binding. You can use any variable name you wish, such as `$index`, `$ind`, `$j`, and many more.

The following PHP script,

```
<?php  
$grades = array("B+", "A+", "A", "C-");  
for ($i = 0; $i <= 3; $i++) {  
    process $grades[$i];  
}  
?>
```

“processes” all elements of the array `$grades`, one at each iteration.

Notice: Please note that since array `$grades` contains four elements, the counted loop structure should iterate from 0 to 3 and not from 1 to 4. This is because the four elements exist in positions 0, 1, 2, and 3.

Let's see some real examples. The following code fragment lets the user enter 100 values to the array `$a`.

```
$a = array();
for ($i = 0; $i <= 99; $i++) {
    $a[$i] = trim(fgets(STDIN));
}
```

The following code fragment doubles the values of all elements of the array `$a`.

```
$a = array(80, 65, 60, 72, 40, 30);
for ($i = 0; $i <= 5; $i++) {
    $a[$i] = $a[$i] * 2;
}
```

The following code fragment displays all values of the array `$a`.

```
$a = array();
for ($i = 0; $i <= 99; $i++) {
    echo $a[$i], "\t";
}
```

Exercise 33.4-1 *Displaying Words in Reverse Order*

Write a PHP script that lets the user enter ten words and then displays them in reverse order.

Solution

At first glance you might be tempted not to use an array. One possible solution would be to use ten individual variables, as follows.

```
<?php
//Input
$word1 = trim(fgets(STDIN));
$word2 = trim(fgets(STDIN));
$word3 = trim(fgets(STDIN));
...
$word10 = trim(fgets(STDIN));

//Output
echo $word10;
echo $word9;
echo $word8;
```

```
...  
echo $word1;  
?>
```

If you are really satisfied with this programming style, go ahead. However, what if the wording of this exercise asked the user to enter 1000 instead of 10 words? Think about it!

Arrays are especially perfect for problems like this one. The following is an appropriate solution.

file_33.4-1.php

```
<?php  
$words = array();  
for ($i = 0; $i <= 9; $i++) {  
    $words[$i] = trim(fgets(STDIN));  
}  
  
for ($i = 9; $i >= 0; $i--) {  
    echo $words[$i];  
}  
?>
```

Remember! Since index numbering starts at zero, the index of the last element of an array is 1 less than the total number of elements in the array.

Notice: Sometimes, the wording of an exercise may say nothing about using an array. However, this doesn't mean that you can't use one. Use arrays whenever you think they are necessary.

Exercise 33.4-2 *Displaying Positive Numbers in Reverse Order*

Write a PHP script that lets the user enter 100 numbers and then displays only the positive ones in reverse order.

Solution

In this exercise, the script should accept all values from the user and store them in an array. However, within the counted loop structure that is responsible for displaying the elements of the array, a nested decision control structure should check for and display only the positive values. The solution is as follows.

file_33.4-2.php

```
<?php  
define("ELEMENTS", 100);  
  
$values = array();
```

```

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $values[$i] = trim(fgets(STDIN));
}

for ($i = ELEMENTS - 1; $i >= 0; $i--) {
    if ($values[$i] > 0) {
        echo $values[$i];
    }
}

?>

```

Notice: A very good tactic for dealing with array sizes is to use constants.

Remember! Since index numbering starts at zero, the index of the last element of an array is 1 less than the total number of elements in the array.

Exercise 33.4-3 *Displaying Even Numbers in Odd-Numbered Index Positions*

Write a PHP script that lets the user enter 100 integers into an array and then displays any even values that are stored in odd-numbered index positions.

Solution

Following is one possible solution.

file_33.4-3a.php

```

<?php
define("ELEMENTS", 100);

$values = array();

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $values[$i] = trim(fgets(STDIN));
}

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($i % 2 != 0 && $values[$i] % 2 == 0) {
        echo $values[$i];
    }
}

?>

```

However, you know that only the values in odd-numbered index positions should be examined. Therefore, in the counted loop structure that is responsible for displaying the elements of the array, instead of starting counting from 0 and using an *offset* of

+1, you can start counting from 1 and use an *offset* of +2. This modification decreases the number of iterations by half. The modified PHP script follows.

file_33.4-3b.php

```
<?php
define("ELEMENTS", 100);

$values = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $values[$i] = trim(fgets(STDIN));
}

for ($i = 1; $i <= ELEMENTS - 1; $i += 2) {
    if ($values[$i] % 2 == 0) {
        echo $values[$i];
    }
}
?>
```

Exercise 33.4-4 *Finding the Sum*

Write a PHP script that lets the user enter 50 numbers into an array and then calculates and displays their sum.

Solution

The solution is as follows:

file_33.4-4a.php

```
<?php
define("ELEMENTS", 50);

$values = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $values[$i] = trim(fgets(STDIN));
}

$sum = 0;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $sum = $sum + $values[$i];
}

echo $sum;
```

```
?>
```

If you are wondering if this exercise could have been solved using just one counted loop structure, the answer is “yes.” An alternative solution is presented next.

```
file_33.4-4b.php
```

```
<?php
define("ELEMENTS", 50);

$values = array();
$sum = 0;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $values[$i] = trim(fgets(STDIN));
    $sum = $sum + $values[$i];
}

echo $sum;

?>
```

Let’s clarify something! Even though many processes can be performed inside just one loop structure, it is simpler to carry out each individual process in a separate loop structure. This is probably not so efficient but, since you are still a novice programmer, try to adopt this programming style just for now. Later, when you have the experience and become a PHP guru, you will be able to “merge” many processes in just one loop control structure.

33.5 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The next statement contains a syntax error.

```
$student names = array();
```

2. In a PHP script, two arrays cannot have the same name.
3. In a PHP script, two arrays cannot have the same number of elements.
4. You cannot use a variable as an index in an array.
5. You can use a mathematical expression as an index in an array.
6. If you use a variable as an index in an array, this variable should contain an integer value.
7. In order to calculate the sum of 20 numeric values, you must use an array.
8. You can let the user enter a value into array `$a` using the statement `$a[] = trim(fgets(STDIN));`
9. The following code fragment creates a one-dimensional array of two elements.

```
$names = array();
$names[0] = "John";
```



```
$names[1] = "George";  
$names[2] = "Sally";
```

10. The following code fragment adds the value 10 to index position 7.

```
$values[5] = 7;  
$values[$values[5]] = 10;
```

11. The following code fragment adds the name “Sally” to index position 3.

```
$names = array();  
$names[] = "John";  
$names[] = "George";  
$names[] = "Sally";
```

12. The following statement adds the name “Sally” to index position 2.

```
$names = array("John", "George", "Sally");
```

13. The following code fragment displays the name “Sally” on the screen.

```
$names = array();  
$k = 0;  
$names[$k] = "John";  
$k++;  
$names[$k] = "George";  
$k++;  
$names[$k] = "Sally";  
$k--;  
echo $names[$k];
```

14. The following code fragment is syntactically correct.

```
$names = array();  
$names[] = "John";  
$names[] = "George";  
$names[] = "Sally";  
echo $names[];
```

15. The following code fragment displays the name “Maria” on the screen.

```
$names = array("John", "George", "Sally", "Maria");  
echo $names[intval(pi())];
```

16. The following code fragment satisfies the property of definiteness.

```
$grades = array("B+", "A+", "A", "C-");  
echo $grades[4];
```

17. The following code fragment satisfies the property of definiteness.

```
$values = array(1, 3, 2, 0);  
echo $values[$values[$values[0]]];
```

18. The following code fragment displays the value of 1 on the screen.

```
$values = array(1, 3, 2, 0);  
echo $values[$values[$values[$values[0]]]];
```

19. The following code fragment displays all the elements of the array.

```
$names = array("John", "George", "Sally", "Maria");  
for ($i = 1; $i <= 4; $i++) {  
    echo $names[$i];  
}
```

20. The following code fragment satisfies the property of definiteness.

```
$names = array("John", "George", "Sally", "Maria");  
for ($i = 2; $i <= 4; $i++) {  
    echo $names[$i];  
}
```

21. The following code fragment lets the user enter 100 values to array \$a.

```
for ($i = 0; $i <= 99; $i++) {  
    $a[] = trim(fgets(STDIN));  
}
```

22. If array \$a contains 30 elements, the following code fragment doubles the values of all of its elements.

```
for ($i = 29; $i >= 0; $i--) {  
    $a[$i] = $a[$i] * 2;  
}
```

23. If array \$a contains 30 elements, the following code fragment displays all of them.

```
for ($i = 0; $i <= 29; $i += 2) {  
    echo $a[$i];  
}
```

33.6 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The following statement

```
$last names = array[]
```

- contains a logic error.
- contains a syntax error.
- contains two syntax errors.

d. contains three syntax errors.

2. The following code fragment

```
$x = 5;  
$names[$x / 2] = 10;
```

- a. does not satisfy the property of definiteness.
- b. does not satisfy the property of finiteness.
- c. does not satisfy the property of effectiveness.
- d. none of the above

3. If variable `$x` contains the value 4, the following statement

```
$names[$x + 1] = 5
```

- a. assigns the value 4 to index position 5.
- b. assigns the value 5 to index position 4.
- c. assigns the value 5 to index position 5.
- d. none of the above

4. The following code fragment

```
$names = array();  
$names[] = 5;
```

- a. assigns the value 5 to index position 1.
- b. assigns the value 5 to index position 0.
- c. does not satisfy the property of definiteness.
- d. none of the above

5. The following code fragment

```
$values[0] = 1;  
$values[$values[0]] = 2;  
$values[$values[1]] = 3;  
$values[$values[2]] = 4;
```

- a. assigns the value 4 to index position 3.
- b. assigns the value 3 to index position 2.
- c. assigns the value 2 to index position 1.
- d. all of the above
- e. none of the above

6. If the array `$values` contains numeric values, the following statement

```
echo $values[$values[1] - $values[1 % 2]] - $values[intval(1 / 2)];
```

- a. does not satisfy the property of definiteness.
- b. always displays 0.
- c. always displays 1.
- d. none of the above

7. You can iterate through a one-dimensional array with a counted loop structure that uses
- variable `$i` as a counter.
 - variable `$j` as a counter.
 - variable `$k` as a counter.
 - any variable as a counter.
8. The following code fragment

```
$names = array("George", "John", "Maria", "Sally");  
for ($i = 3; $i >= 1; $i--) {  
    echo $names[$i];  
}
```

- displays all names in ascending order.
 - displays some names in ascending order.
 - displays all names in descending order.
 - displays some names in descending order.
 - none of the above
9. If array `$a` contains 30 elements, the following code fragment

```
for ($i = 29; $i >= 1; $i--) {  
    $a[$i] = $a[$i] * 2;  
}
```

- doubles the values of some of its elements.
- doubles the values of all of its elements.
- none of the above

33.7 Review Exercises

Complete the following exercises.

1. Create the trace table for the following code fragment. At the end, the array should contain three elements.

```
$a = array();  
$a[2] = 1;  
$x = 0;  
$a[$x + $a[2]] = 4;  
$a[$x] = $a[$x + 1] * 4;
```

2. Create the trace table for the following code fragment. At the end, the array should contain five elements.

```
$a = array();  
$a[1] = 5;  
$x = 0;  
$a[$x] = 4;
```

```
$a[$a[0]] = $a[$x + 1] % 3;
$a[$a[0] / 2] = 10;
$x += 2;
$a[$x + 1] = $a[$x] + 9;
```

3. Create the trace table for the following code fragment for three different executions. At the end, the array should contain four elements.

The input values for the three executions are: (i) 3, (ii) 4, and (iii) 1.

```
$a = array();
$a[1] = trim(fgets(STDIN));
$x = 0;
$a[$x] = 3;
$a[$a[0]] = $a[$x + 1] % 2;
$a[$a[0] % 2] = 10;
$x++;
$a[$x + 1] = $a[$x] + 9;
```

4. Create the trace table for the following code fragment for three different executions. At the end, the array should contain four elements.

The input values for the three executions are: (i) 100, (ii) 108, and (iii) 1.

```
$a = array();
$a[1] = trim(fgets(STDIN));
$x = 0;
$a[$x] = 3;
$a[$a[0]] = $a[$x + 1] % 10;
if ($a[3] > 5) {
    $a[$a[0] % 2] = 9;
    $x += 1;
    $a[$x + 1] = $a[$x] + 9;
}
else {
    $a[2] = 3;
}
```

5. Fill in the gaps in the following trace table. In steps 6 and 7, fill in the name of a variable; for all other cases, fill in constant values, arithmetic, or comparison operators.

Step	Statement	\$x	\$y	\$a[0]	\$a[1]	\$a[2]
1	\$a = array()	?	?	?	?	?
2	\$x = ...	4	?	?	?	?
3	\$y = \$x -	4	3	?	?	?

4, 5	if (\$x \$y)					
	\$a[0] = ... ;	4	3	1	?	?
	else					
	\$a[0] = \$y;					
6	\$a[1] = ... + 3	4	3	1	7	?
7	\$y = ... - 1	4	2	1	7	?
8	\$a[\$y] = (\$x + 5) 2	4	2	1	7	1

6. Create the trace table for the following code fragment.

```
$a = array(17, 12, 45, 12, 12, 49);

for ($i = 0; $i <= 5; $i++) {
    if ($a[$i] == 12)
        $a[$i]--;
    else
        $a[$i]++;
}
```

7. Create the trace table for the following code fragment.

```
$a = array(10, 15, 12, 23, 22, 19);

for ($i = 1; $i <= 4; $i++) {
    $a[$i] = $a[$i + 1] + $a[$i - 1];
}
```

- Write a PHP script that lets the user enter 100 numbers in an array and then displays these values raised to the power of three.
- Write a PHP script that lets the user enter 80 numbers in an array and then displays these values in reverse order raised to the power of two.
- Write a PHP script that lets the user enter 90 integers in an array and then displays in reverse order those that are exactly divisible by 5.
- Write a PHP script that lets the user enter 50 integers in an array and then displays those that are even or greater than 10.
- Write a PHP script that lets the user enter 30 numbers in an array and then calculates and displays the sum of those that are positive.
- Write a PHP script that lets the user enter 50 integers in an array and then calculates and displays the sum of those that have two digits.
Hint: All two-digit integers are between 10 and 99.
- Write a PHP script that lets the user enter 40 numbers in an array and then calculates and displays the sum of the positive numbers and the sum of the negative ones.

15. Write a PHP script that lets the user enter 20 numbers in an array and then calculates and displays their average value.
16. Write a PHP script that lets the user enter 50 words and displays those that contain at least 10 characters.

Hint: Use the `strlen()` function.

17. Write a PHP script that lets the user enter 40 words and displays those that contain the letter “w” at least twice.

Hint: Use the `substr()` and `strlen()` functions.

Chapter 34

Two-Dimensional Arrays

34.1 Creating Two-Dimensional Arrays in PHP

An array that can hold the grades of four lessons for three students is as follows.

\$grades =

	Lessons				
	0	1	2	3	
0	B+	A+	A	C-	Students
1	B	A	C	A+	
2	B	B	B-	B+	

Notice: A two-dimensional array has rows and columns. In this particular example, array `grades` has 3 rows and 4 columns.

Actually, PHP supports only one-dimensional arrays. But there is a trick that you can use to overcome this problem and create multidimensional arrays. You can create an array of arrays! Just imagine the array `$grades` as a one-column array with three positions, each of which contains a whole new array of four elements, as follows.

\$grades =

	Lessons				
	B+	A+	A	C-	Students
	B	A	C	A+	
	B	B	B-	B+	

Notice! In PHP, a multidimensional array is actually an array of arrays! A two-dimensional array is an array of arrays, a three-dimensional array is an array of arrays of arrays, and so on.

Remember! In Chapter 5 you learned about the rules that must be followed when assigning names to PHP variables. Assigning names to PHP arrays follows exactly the same rules!

As in one-dimensional arrays, there are several approaches to creating and adding values (elements) to a two-dimensional array. Let's try to create the array `$grades` using each of these approaches.

First Approach

You can create an array in PHP using the following statement, given in general form,

```
$array_name = array();
```

and then you can add a whole new array to one of its positions using the following statement, given in general form

```
$array_name[index] = array(value0, value1, value2, ..., valueM);
```

where *index* is the index position in the array.

The following code fragment creates the array `$grades` and adds values to it.

```
$grades = array();  
$grades[0] = array("B+", "A+", "A", "C-");  
$grades[1] = array("B", "A", "C", "A+");  
$grades[2] = array("B", "B", "B-", "B+");
```

Second Approach

You can create an array and add new arrays to its positions without using *index*, as shown in the following code fragment, given in general form.

```
$array_name = array();  
$array_name[] = array(value0-0, value0-1, value0-2, ..., value0-M);  
$array_name[] = array(value1-0, value1-1, value1-2, ..., value1-M);  
$array_name[] = array(value2-0, value2-1, value2-2, ..., value2-M);  
...  
$array_name[] = array(valueN-0, valueN-1, valueN-2, ..., valueN-M);
```

The next code fragment creates the array `$grades` and adds values to it.

```
$grades = array();  
$grades[] = array("B+", "A+", "A", "C-");  
$grades[] = array("B", "A", "C", "A+");  
$grades[] = array("B", "B", "B-", "B+");
```

Notice: Please note that in this approach, index numbering starts at zero by default.

Third Approach

Another way to create an array and directly add values to it is shown in the following PHP statement, given in general form.

```
$array_name = array( array(value0-0, value0-1, value0-2, ..., value0-M),  
                     array(value1-0, value1-1, value1-2, ..., value1-M),  
                     array(value2-0, value2-1, value2-2, ..., value2-M),  
                     ...
```

```
        array(valueN-0, valueN-1, valueN-2, ..., valueN-M)
    );
```

Thus, the array `$grades` can be created using the following statement.

```
$grades = array( array("B+", "A+", "A", "C-"),
                 array("B", "A", "C", "A+"),
                 array("B", "B", "B-", "B+")
               );
```

Fourth Approach

If you wish to create an array and directly add values to certain positions you can use the following PHP statement, given in general form.

```
$array_name = array( index0 => array(value0-0, value0-1, ..., value0-M),
                    index1 => array(value1-0, value1-1, ..., value1-M),
                    index2 => array(value2-0, value2-1, ..., value2-M),
                    ...
                    indexN => array(valueN-0, valueN-1, ..., valueN-M)
                  );
```

Thus, the array `$grades` can be created using the following statement.

```
$grades = array( 0 => array("B+", "A+", "A", "C-"),
                1 => array("B", "A", "C", "A+"),
                2 => array("B", "B", "B-", "B+")
              );
```

In this approach, elements can also be in random order.

```
$grades = array( 2 => array("B", "B", "B-", "B+"),
                1 => array("B", "A", "C", "A+"),
                0 => array("B+", "A+", "A", "C-")
              );
```

Fifth Approach

Last but not least, if you wish to create an array and add values using one statement for each, you can use the following PHP statement, given in general form.

```
$array_name = array(array());
$array_name[0][0] = value0-0;
$array_name[0][1] = value0-1;
$array_name[0][2] = value0-2;
...
$array_name[0][M] = value0-M;

$array_name[1][0] = value1-0;
$array_name[1][1] = value1-1;
```

```

$array_name[1][2] = value1-2;
...
$array_name[1][M] = value1-M;

$array_name[N][0] = valueN-0;
$array_name[N][1] = valueN-1;
$array_name[N][2] = valueN-2;
...
$array_name[N][M] = valueN-M;

```

Thus, the array `$grades` can be created using the following code fragment.

```

$grades = array(array());
$grades[0][0] = "B+";
$grades[0][1] = "A+";
$grades[0][2] = "A";
$grades[0][3] = "C-";
$grades[1][0] = "B";
$grades[1][1] = "A";
$grades[1][2] = "C";
$grades[1][3] = "A+";
$grades[2][0] = "B";
$grades[2][1] = "B";
$grades[2][2] = "B-";
$grades[2][3] = "B+";

```

34.2 How to Get Values from Two-Dimensional Arrays

A two-dimensional array consists of rows and columns. The following example shows a two-dimensional array with three rows and four columns.

	<i>Column 0</i>	<i>Column 1</i>	<i>Column 2</i>	<i>Column 3</i>
<i>Row 0</i>				
<i>Row 1</i>				
<i>Row 2</i>				

Each element of a two-dimensional array can be uniquely identified using a pair of indexes: a row index, and a column index, as shown next.

```

$array_name[row_index][column_index]

```

The following PHP script creates the two-dimensional array `$grades` having three rows and four columns, and then displays some of its elements.

```

<?php
$grades = array();

```

```
$grades[0] = array("B+", "A+", "A", "C-");
$grades[1] = array("B", "A", "C", "A+");
$grades[2] = array("B", "B", "B-", "B+");

echo $grades[1][2];      //This displays C on the screen
echo $grades[2][2];      //This displays B- on the screen
echo $grades[0][0];      //This displays B+ on the screen
?>
```

Exercise 34.2-1 *Creating the Trace Table*

Create the trace table for the next code fragment.

```
$a = array (array(0, 0),
            array(0, 0),
            array(0, 0)
        );

$a[1][0] = 9;
$a[0][1] = 1;
$a[0][0] = $a[0][1] + 6;
$x = 2;
$a[$x][1] = $a[0][0] + 4;
$a[$x - 1][1] = $a[0][1] * 3;
$a[$x][0] = $a[$x - 1][1] - 3;
Solution
```

This code fragment uses a 3 × 2 array, that is, an array that has 3 rows and 2 columns. The trace table is as follows.

Step	Statement	Notes	\$x	\$a						
1	<pre>\$a = array(array(0, 0), array(0, 0), array(0, 0))</pre>	This creates array \$a with zero values in it.	?	<table><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0
0	0									
0	0									
0	0									
2	<pre>\$a[1][0] = 9</pre>		?	<table><tr><td>0</td><td>0</td></tr><tr><td>9</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	0	0	9	0	0	0
0	0									
9	0									
0	0									

3	<code>\$a[0][1] = 1</code>		?	<table><tr><td>0</td><td>1</td></tr><tr><td>9</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	0	1	9	0	0	0
0	1									
9	0									
0	0									
4	<code>\$a[0][0] = \$a[0][1] + 6</code>		?	<table><tr><td>7</td><td>1</td></tr><tr><td>9</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	7	1	9	0	0	0
7	1									
9	0									
0	0									
5	<code>\$x = 2</code>		2	<table><tr><td>7</td><td>1</td></tr><tr><td>9</td><td>0</td></tr><tr><td>0</td><td>0</td></tr></table>	7	1	9	0	0	0
7	1									
9	0									
0	0									
6	<code>\$a[\$x][1] = \$a[0][0] + 4</code>		2	<table><tr><td>7</td><td>1</td></tr><tr><td>9</td><td>0</td></tr><tr><td>0</td><td>11</td></tr></table>	7	1	9	0	0	11
7	1									
9	0									
0	11									
7	<code>\$a[\$x - 1][1] = \$a[0][1] * 3</code>		2	<table><tr><td>7</td><td>1</td></tr><tr><td>9</td><td>3</td></tr><tr><td>0</td><td>11</td></tr></table>	7	1	9	3	0	11
7	1									
9	3									
0	11									
8	<code>\$a[\$x][0] = \$a[\$x - 1][1] - 3</code>		2	<table><tr><td>7</td><td>1</td></tr><tr><td>9</td><td>3</td></tr><tr><td>0</td><td>11</td></tr></table>	7	1	9	3	0	11
7	1									
9	3									
0	11									

34.3 How to Add Values Entered by the User to a Two-Dimensional Array

Just as in one-dimensional arrays, instead of reading a value entered from the keyboard and assigning that value to a variable, you can directly assign that value to a specific position of an array. The next code fragment creates a two-dimensional array `$test`, lets the user enter four values, and adds those values to the array.

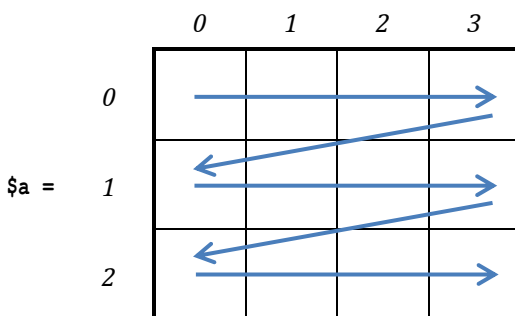
```
$test = array (array());
$test[0][0] = trim(fgets(STDIN));
$test[0][1] = trim(fgets(STDIN));
$test[1][0] = trim(fgets(STDIN));
$test[1][1] = trim(fgets(STDIN));
```

34.4 How to Iterate Through a Two-Dimensional Array

Since a two-dimensional array consists of rows and columns, a script can iterate either through rows or through columns.

Iterating through rows

Iterating through rows means that row 0 is processed first, row 1 is process next, row 2 afterwards, and so on. Next there is an example of a 3×4 array. The arrows show the “path” that is followed when iteration through rows is performed or in other words, they show the order in which the elements are processed.



Remember! A 3×4 array is a two-dimensional array that has 3 rows and 4 columns. In the notation $Y \times X$, the first number (Y) always represents the total number of rows and the second number (X) always represents the total number of columns.

When iterating through rows, the elements of the array are processed as follows:

- the elements of row 0 are processed in the following order
`$a[0][0] → $a[0][1] → $a[0][2] → $a[0][3]`
- the elements of row 1 are processed in the following order
`$a[1][0] → $a[1][1] → $a[1][2] → $a[1][3]`
- the elements of row 2 are processed in the following order
`$a[2][0] → $a[2][1] → $a[2][2] → $a[2][3]`

Using PHP statements, let's try to process all elements of a 3×4 array (3 rows $\times$ 4 columns) iterating through rows.

```
$i = 0;                                     //Variable $i refers to Row 0.
for ($j = 0; $j <= 3; $j++) {              //This loop control structure processes all
    process $a[$i][$j];                     //elements of Row 0
}

$i = 1;                                     //Variable $i refers to Row 1.
for ($j = 0; $j <= 3; $j++) {              //This loop control structure processes all
    process $a[$i][$j];                     //elements of Row 1
}
```

```
$i = 2;                                     //Variable $i refers to Row 2.
for ($j = 0; $j <= 3; $j++) {             //This loop control structure processes all
    process $a[$i][$j];                     //elements of Row 2
}
```

Of course, the same results can be achieved using a nested loop control structure as shown next.

```
for ($i = 0; $i <= 2; $i++) {
    for ($j = 0; $j <= 3; $j++) {
        process $a[$i][$j];
    }
}
```

Let's see some examples. The following code fragment lets the user enter $10 \times 10 = 100$ values to array \$a.

```
for ($i = 0; $i <= 9; $i++) {
    for ($j = 0; $j <= 9; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}
```

The following code fragment decreases all values of array \$a by one.

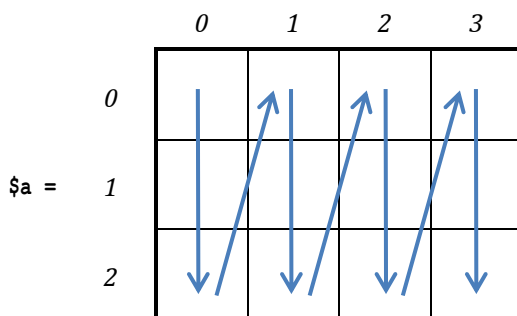
```
for ($i = 0; $i <= 9; $i++) {
    for ($j = 0; $j <= 9; $j++) {
        $a[$i][$j] = $a[$i][$j] - 1;
    }
}
```

The following code fragment displays the values of array \$a.

```
for ($i = 0; $i <= 9; $i++) {
    for ($j = 0; $j <= 9; $j++) {
        echo $a[$i][$j], "\t";
    }
    echo "\n";
}
```

Iterating Through Columns

Iterating through columns means that column 0 is processed first, column 1 is processed next, column 2 afterwards, and so on. Next there is an example of a 3×4 array. The arrows show the order in which the elements are processed.



When iterating through columns, the elements of the array are processed as follows:

- the elements of column 0 are processed in the following order

`$a[0][0] → $a[1][0] → $a[2][0]`

- the elements of column 1 are processed in the following order

`$a[0][1] → $a[1][1] → $a[2][1]`

- the elements of column 2 are processed in the following order

`$a[0][2] → $a[1][2] → $a[2][2]`

- the elements of column 3 are processed in the following order

`$a[0][3] → $a[1][3] → $a[2][3]`

Using PHP statements, let's try to process all elements of a 3×4 array (3 rows $\times$ 4 columns) by iterating through columns.

```
$j = 0; //Variable $j refers to Column 0.
for ($i = 0; $i <= 2; $i++) { //This loop control structure processes all
    process $a[$i][$j]; //elements of Column 0
}

$j = 1; //Variable $j refers to Column 1.
for ($i = 0; $i <= 2; $i++) { //This loop control structure processes all
    process $a[$i][$j]; //elements of Column 1
}

$j = 2; //Variable $j refers to Column 2.
for ($i = 0; $i <= 2; $i++) { //This loop control structure processes all
    process $a[$i][$j]; //elements of Column 2
}

$j = 3; //Variable $j refers to Column 3.
for ($i = 0; $i <= 2; $i++) { //This loop control structure processes all
    process $a[$i][$j]; //elements of Column 3
}
```

Of course, the same result can be achieved using a nested loop control structure as shown next.

```
for ($j = 0; $j <= 3; $j++) {  
    for ($i = 0; $i <= 2; $i++) {  
        process $a[$i][$j];  
    }  
}
```

As you can see, this code fragment differs on only one single point from the one that iterates through rows: the two counted loop structures have switched places. Be careful though. Don't try switching places of the two index variables `$i` and `$j` in the statement `process $a[$i][$j]`. Take the following code fragment, for example. It tries to iterate through columns in a 3×4 array (3 rows $\times$ 4 columns) but it does not satisfy the property of definiteness. Can you find out why?

```
for ($j = 0; $j <= 3; $j++) {  
    for ($i = 0; $i <= 2; $i++) {  
        process $a[$j][$i];  
    }  
}
```

The trouble arises when variable `$j` becomes equal to 3. The statement `process $a[$j][$i]` tries to process the elements at **row** index 3 (this is the fourth row) which, of course, does not exist! Still confused? Don't be! There is no row index 3 in a 3×4 array! Why? Since row index numbering starts at 0, only rows 0, 1, and 2 actually exist!

Exercise 34.4-1 *Displaying Reals Only*

Write a PHP script that prompts the user to enter numeric values in a 5×7 array and then displays the indexes of the positions that contain reals.

Solution

Iterating through rows is the most popular approach, so let's use it. The solution is as follows.

`file_34.4-1.php`

```
<?php  
define("ROWS", 5);  
define("COLUMNS", 7);  
  
$a = array(array());  
for ($i = 0; $i <= ROWS - 1; $i++) {  
    for ($j = 0; $j <= COLUMNS - 1; $j++) {  
        echo "Enter a value for element ", $i , " , " , $j, " : ";
```

```

        $a[$i][$j] = trim(fgets(STDIN));
    }
}

for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        if ($a[$i][$j] != intval($a[$i][$j])) {
            echo "A real found at position: ", $i , " , ", $j, "\n";
        }
    }
}
?>

```

Exercise 34.4-2 *Displaying Odd Columns Only*

Write a PHP script that prompts the user to enter numeric values in a 5×7 array and then displays the elements of the columns with odd-numbered indexes (that is, column indexes 1, 3, and 5).

Solution

The PHP script is presented next.

file_34.4-2.php

```

<?php
define("ROWS", 5);
define("COLUMNS", 7);

$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        echo "Enter a value for element ", $i , " , ", $j, ": ";
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

//Iterate through columns
for ($j = 1; $j <= COLUMNS - 1; $j += 2) {    //Start from 1 and
                                                //increment by 2

    for ($i = 0; $i <= ROWS - 1; $i++) {
        echo $a[$i][$j], " ";
    }
}

```

```
?>
```

Notice: This book tries to use, as often as possible, variable `$i` as the row index and variable `$j` as the column index. Of course, you can use other variable names as well, such as `$row`, `$r` for row index, or `$column`, `$c` for column index, but variables `$i` and `$j` are widely used by the majority of programmers. After using them for a while, your brain will relate `$i` to rows and `$j` to columns. Thus, every algorithm or script that uses these variable names as indexes in two-dimensional arrays will be more readily understood.

34.5 What’s the Story on Variables `$i` and `$j`?

Many programmers believe that the name of variable `$i` stands for “index” and `$j` is used just because it is after `$i`. Others believe that the name `$i` stands for “integer.” Probably the truth lies somewhere in the middle.

Mathematicians were using `i`, `j`, and `k` to designate integers in mathematics long before computers were around. Later, in FORTRAN, one of the first high-level computer languages, variables `i`, `j`, `k`, `l`, `m`, and `n` were integers by default. Thus, the first programmers picked up the habit of using variables `i` and `j` in their programs and it became a convention in most computer languages.

34.6 Square Arrays

An array that has the same number of rows and columns is called a square array. Following are some examples of square arrays.

	0	1	2
0			
1			
2			

	0	1	2	3	4
0					
1					
2					
3					
4					

Exercise 34.6-1 Finding the Sum of the Elements of the Main Diagonal

Write a PHP script that lets the user enter numeric values into a 10 × 10 square array and then calculates the sum of the elements of its main diagonal.

Solution

The main diagonal of a square array is the collection of those elements that runs from the top left corner to the bottom right corner. Following are some examples of square arrays with their main diagonals highlighted by a dark background.

	0	1	2
0	10	12	11
1	23	50	9
2	12	11	-3

	0	1	2	3	4
0	-3	44	-12	25	22
1	10	-1	29	12	-9
2	5	-3	16	22	-8
3	11	25	12	25	-5
4	12	22	53	44	-15

Notice: Please note that the elements of the main diagonal have their row index equal to their column index.

You can calculate the sum of the elements of the main diagonal using two different approaches. Let's study them both.

First Approach – Iterating through all elements

In this approach, the script iterates through rows, or even through columns, and checks if the row index is equal to the column index. For square arrays, represented as $N \times N$, the number of rows and columns is equal, so you can define just one constant, N . The solution is as follows.

file_34.6-1a.php

```
<?php
define("N", 10);

$a = array(array());
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

//Calculate Sum
$sum = 0;
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        if ($i == $j) {
            $sum = $sum + $a[$i][$j];
        }
    }
}
```

```
echo "Sum = ", $sum;
?>
```

Notice: Please note that the script iterates through rows and checks if the row index is equal to the column index. Alternatively, the same result can be achieved by iterating through columns.

Notice: In this approach, the nested loop control structure that is responsible for calculating the sum performs $10 \times 10 = 100$ iterations.

Second Approach – Iterating directly through the main diagonal

In this approach, one single loop control structure iterates directly through the main diagonal. The solution is as follows.

file_34.6-1b.php

```
<?php
define("N", 10);

$a = array(array());
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

//Calculate Sum
$sum = 0;
for ($k = 0; $k <= N - 1; $k++) {
    $sum = $sum + $a[$k][$k];
}

echo "Sum = ", $sum;
?>
```

Notice: This approach is much more efficient than the first one since the total number of iterations performed by the counted loop structure that is responsible for calculating the sum is just 10.

Exercise 34.6-2 Finding the Sum of the Elements of the Antidiagonal

Write a PHP script that lets the user enter numeric values in a 5×5 square array and then calculates the sum of the elements of its antidiagonal.

Solution

The antidiagonal of a square array is the collection of those elements that runs from the top right corner to the bottom left corner of the array. Next, you can find an example of a 5 × 5 square array with its antidiagonal highlighted by a dark background.

		0	1	2	3	4
\$a =	0	-3	44	-12	25	22
	1	10	-1	29	12	-9
	2	5	-3	16	22	-8
	3	11	25	12	25	-5
	4	12	22	53	44	-15

Any element of the antidiagonal of an N × N array satisfies the following equation:

$$i + j = N - 1$$

where variables *i* and *j* correspond to the row and column indexes of the element respectively.

If you solve for *j*, the equation becomes

$$j = N - i - 1$$

Using this formula, you can calculate, for any value of variable *i*, the corresponding value of variable *j*. For example, in the previous 5 × 5 square array in which N equals 5, when *i* is 0 the value of variable *j* is

$$j = N - i - 1 \Leftrightarrow j = 5 - 0 - 1 = \Leftrightarrow j = 4$$

Doing this for all values of variable *i*, you can calculate the corresponding values of variable *j* as shown in the following table.

Value of Variable <i>i</i>	Calculated Value of Variable <i>j</i>
0	4
1	3
2	2
3	1
4	0

Using all this knowledge, let’s now write the corresponding PHP script.

file_34.6-2.php

```
<?php
define("N", 5);

$a = array(array());
```

```

for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

//Calculate Sum
$sum = 0;
for ($i = 0; $i <= N - 1; $i++) {
    $j = N - $i - 1;
    $sum = $sum + $a[$i][$j];
}

echo "Sum = ", $sum;
?>

```

Notice: Please note that the counted loop structure that is responsible for finding the sum of the elements of the antidiagonal iterates directly through the antidiagonal.

Exercise 34.6-3 Filling in the Array

Write a PHP script that creates and displays the following array.

	0	1	2	3	4
0	-1	20	20	20	20
1	10	-1	20	20	20
2	10	10	-1	20	20
3	10	10	10	-1	20
4	10	10	10	10	-1

\$a =

Solution

As you can see, in the main diagonal, there is the value of -1. You already know that the main diagonal of a square array is the collection of those elements that have their row index equal to their column index. Now, what you also need is to find a common characteristic between all positions that contain the value 10, and another such common characteristic between all positions that contain the value 20.

If you look closer, you will realize that the row index of any position that contains the value 10 is always greater than its corresponding column index and, similarly, the row index of any position that contains the value 20 is always less than its corresponding column index.

Accordingly, the PHP script is as follows.

file_34.6-3.php

```
<?php
define("N", 5);

$a = array(array());
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        if ($i == $j) {
            $a[$i][$j] = -1;
        }
        elseif ($i > $j) {
            $a[$i][$j] = 10;
        }
        else {
            $a[$i][$j] = 20;
        }
    }
}

for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        echo $a[$i][$j], "\t";
    }
    echo "\n";
}

?>
```

34.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. All the positions of a two-dimensional array must contain different values.
2. In order to refer to a position of a two-dimensional array you need two indexes.
3. The two indexes of a two-dimensional array must be either both variables, or both constant values.
4. A 5×6 array is a two-dimensional array that has five columns and six rows.
5. To refer to an element of array `$a` that exists at the second row and third column, you would write `$a[2][3]`.
6. Iterating through rows means that first row of a two-dimensional array is processed first, the second row is process next, and so on.

7. You cannot use variables other than `$i` and `$j` to iterate through a two-dimensional array.
8. The following PHP statement creates a two-dimensional array of no elements.

```
$names = array(array());
```

9. The following code fragment creates a two-dimensional array of four elements.

```
$names = array(array());  
$names[0][0] = "John";  
$names[0][1] = "George";  
$names[1][0] = "Sally";  
$names[1][1] = "Angelina";
```

10. The following code fragment adds the value 10 to the row with index 0.

```
$values[0][0] = 7;  
$values[0][$values[0][0]] = 10;
```

11. The following statement adds the name "Sally" to the row with index 1.

```
$names = array(array("John", "George"),  
                array("Sally", "Angelina"));
```

12. The following code fragment displays the name "Sally" on the screen.

```
$names = array();  
$k = 0;  
$names[0][$k] = "John";  
$k++;  
$names[0][$k] = "George";  
$names[1][$k] = "Sally";  
$k--;  
$names[1][$k] = "Angelina";  
echo $names[1][1];
```

13. The following code fragment satisfies the property of definiteness.

```
$grades = array(array("B+", "A+"),  
                array("A", "C-"));  
echo $grades[2][2];
```

14. The following code fragment satisfies the property of definiteness.

```
$values = array(array(1, 0),  
                array(2, 0));  
echo $values[$values[0][0]][$values[0][1]];
```

15. The following code fragment displays the value 2 on the screen.

```
$values = array(array(0, 1),
```

```
        array(2, 0));  
echo $values[$values[0][1]][$values[0][0]];
```

16. The following code fragment displays all the elements of a 3×4 array.

```
for ($k = 0; $k <= 11; $k++) {  
    $i = intval($k / 4);  
    $j = $k % 4;  
    echo $names[$i][$j];  
}
```

17. The following code fragment lets the user enter 100 values to array \$a.

```
for ($i = 0; $i <= 9; $i++) {  
    for ($j = 0; $j <= 9; $j++) {  
        $a[$i][$j] = trim(fgets(STDIN));  
    }  
}
```

18. If array \$a contains 10×20 elements, the following code fragment doubles the values of all of its elements.

```
for ($i = 9; $i >= 0; $i--) {  
    for ($j = 19; $j >= 0; $j--) {  
        $a[$i][$j] *= 2;  
    }  
}
```

19. If array \$a contains 10×20 elements, the following code fragment displays some of them.

```
for ($i = 0; $i <= 8; $i += 2) {  
    for ($j = 0; $j <= 19; $j++) {  
        echo $a[$i][$j];  
    }  
}  
  
for ($i = 1; $i <= 9; $i += 2) {  
    for ($j = 0; $j <= 19; $j++) {  
        echo $a[$i][$j];  
    }  
}
```

20. The following code fragment displays only the columns with even-numbered indexes.

```
for ($j = 0; $j <= 10; $j += 2) {  
    for ($i = 0; $i <= 9; $i++) {  
        echo $a[$i][$j];  
    }  
}
```

21. A 5×5 array is a square array.
22. In the main diagonal of a square array, all elements have their row index equal to their column index.
23. The antidiagonal of a square array is the collection of those elements that runs from the top left corner to the bottom right corner of the array.
24. Any element of the antidiagonal of an $N \times N$ array satisfies the equation $i + j = N - 1$, where variables i and j correspond to the row and column indexes of the element respectively.
25. The following code fragment calculates the sum of the elements of the main diagonal of a $N \times N$ array.

```
$sum = 0;
for ($k = 0; $k <= N - 1; $k++) {
    $sum += $a[$k][$k];
}
```

26. The following code fragment displays all the elements of the antidiagonal of an $N \times N$ array.

```
for ($i = N - 1; $i >= 0; $i--) {
    echo $a[$i][N - $i - 1];
}
```

27. The column index of any element of a square array that is below the main diagonal is always greater than its corresponding row index.

34.8 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The following statement

```
$last_names[] = array(array());
```

- a. contains a logic error.
- b. contains a syntax error.
- c. contains two syntax errors.
- d. none of the above

2. The following code fragment

```
$values = array(array(1, 0)
                 array(2, 0));
echo $values[$values[0][0], $values[0][1]];
```

- a. contains a logic error.
- b. contains a syntax error.
- c. contains two syntax errors.
- d. none of the above

3. The following code fragment

```
$x = trim(fgets(STDIN));  
$y = trim(fgets(STDIN));  
$names[$x][$y] = 10;
```

- a. does not satisfy the property of finiteness.
- b. does not satisfy the property of effectiveness.
- c. does not satisfy the property of definiteness.
- d. none of the above

4. If variable `$x` contains the value 4, the following statement

```
$names[$x + 1][$x] = 5
```

- a. assigns the value 5 to the position with row index 5 and column index 4.
- b. assigns the value 5 to the position with row index 4 and column index 5.
- c. assigns the value 5 to the position with row index 5 and column index 5.
- d. none of the above

5. The following code fragment

```
$names = array();  
$names[] = array(3, 5, 2);
```

- a. assigns the value 5 to the position with row index 0 and column index 1.
- b. assigns the value 3 to the position with row index 0 and column index 0.
- c. assigns the value 2 to the position with row index 0 and column index 2.
- d. all of the above
- e. none of the above

6. The following code fragment

```
$values = array();  
$values[0][0] = 1;  
$values[0][$values[0][0]] = 2;
```

- a. creates a 1×2 array.
- b. creates a 2×1 array.
- c. creates a 1×3 array.
- d. none of the above

7. You can iterate through a two-dimensional array with two nested loop control structures that use

- a. variables `$i` and `$j` as counters.
- b. variables `$k` and `$l` as counters.
- c. variables `$m` and `$n` as counters.
- d. any variables as counters.

8. The following code fragment

```
$names = array(array("John", "Sally"),
               array("George", "Maria"));

for ($j = 0; $j <= 1; $j++) {
    for ($i = 1; $i >= 0; $i--) {
        echo $names[$i][$j];
    }
}
```

- a. displays all names in descending order.
- b. displays some names in descending order.
- c. displays all names in ascending order.
- d. displays some names in ascending order.
- e. none of the above

9. If array `$a` contains 30×40 elements, the following code fragment

```
for ($i = 30; $i >= 1; $i--) {
    for ($j = 40; $j >= 1; $j--) {
        $a[$i][$j] *= 3;
    }
}
```

- a. triples the values of some of its elements.
- b. triples the values of all of its elements.
- c. none of the above

10. If array `$a` contains 30×40 elements, the following code fragment

```
$sum = 0;
for ($i = 29; $i >= 0; $i--) {
    for ($j = 39; $j >= 0; $j--) {
        $sum += $a[$i][$j];
    }
}

$average = $sum / 120;
```

- a. calculates the sum of all of its elements.
- b. calculates the average value of all of its elements.

c. all of the above

11. The following two code fragments calculate the sum of the elements of the main diagonal of an $N \times N$ array,

```
$sum = 0;
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        if ($i == $j) {
            $sum = $sum + $a[$i][$j];
        }
    }
}
```

```
$sum = 0;
for ($k = 0; $k <= N - 1; $k++) {
    $sum = $sum + $a[$k][$k];
}
```

- but the first one is more efficient.
- but the second one is more efficient.
- none of the above; both code fragments perform equivalently

34.9 Review Exercises

Complete the following exercises.

- Create the trace table for the following code fragment. At the end, the array should contain 2×3 elements.

```
$a = array(array());
$a[0][2] = 1;
$x = 0;
$a[0][$x] = 9;
$a[0][$x + $a[0][2]] = 4;
$a[$a[0][2]][2] = 19;
$a[$a[0][2]][$x + 1] = 13;
$a[$a[0][2]][$x] = 15;
```

- Create the trace table for the following code fragment. At the end, the array should contain 2×3 elements.

```
$a = array(array());
for ($i = 0; $i <= 1; $i++) {
    for ($j = 0; $j <= 2; $j++) {
        $a[$i][$j] = ($i + 1) * 5 + $j;
    }
}
```

- Create the trace table for the following code fragment. At the end, the array should contain 3×3 elements.

```
$a = array(array());
```

```
for ($j = 0; $j <= 2; $j++) {  
    for ($i = 0; $i <= 2; $i++) {  
        $a[$i][$j] = ($i + 1) * 2 + $j * 4;  
    }  
}
```

4. Try, without using a trace table, to determine the values that the array will contain when the following code fragment is executed. Do this for three different executions. The corresponding input values are: (i) 5, (ii) 9, and (iii) 3.

```
$a = array(array());  
$x = trim(fgets(STDIN));  
for ($i = 0; $i <= 1; $i++) {  
    for ($j = 0; $j <= 2; $j++) {  
        $a[$i][$j] = ($x + $i) * $j;  
    }  
}
```

5. Try, without using a trace table, to determine the values that the array will contain when the following code fragment is executed. Do this for three different executions. The corresponding input values are: (i) 13, (ii) 10, and (iii) 8.

```
$a = array(array());  
$x = trim(fgets(STDIN));  
for ($i = 0; $i <= 1; $i++) {  
    for ($j = 0; $j <= 2; $j++) {  
        if ($j < $x % 4)  
            $a[$i][$j] = ($x + $i) * $j;  
        else  
            $a[$i][$j] = ($x + $j) * $i + 3;  
    }  
}
```

6. Try, without using a trace table, to determine the values that the array will contain when the following code fragment is executed.

```
$a = array(array(18, 10, 35),  
            array(32, 12, 19));  
  
for ($j = 0; $j <= 2; $j++) {  
    for ($i = 0; $i <= 1; $i++) {  
        if ($a[$i][$j] < 13)  
            $a[$i][$j] /= 2;  
        elseif ($a[$i][$j] < 20)
```



```

        $a[$i][$j]++;
    else
        $a[$i][$j] -= 4;
    }
}

```

7. Try, without using a trace table, to determine the values that the array will contain when the following code fragment is executed.

```

$a = array(array(11, 10),
            array(15, 19),
            array(22, 15));

for ($j = 0; $j <= 1; $j++) {
    for ($i = 0; $i <= 2; $i++) {
        if ($i == 2)
            $a[$i][$j] += $a[$i - 1][$j];
        else
            $a[$i][$j] += $a[$i + 1][$j];
    }
}

```

8. Assume that array \$a contains the following values.

	0	1	2	3
0	-1	15	22	3
1	25	12	16	14
2	7	9	1	45
3	40	17	11	13

What displays on the screen after executing each of the following code fragments?

i)

```

for ($i = 0; $i <= 2; $i++) {
    for ($j = 0; $j <= 2; $j++) {
        echo $a[$i][$j], " ";
    }
}

```

ii)

```

for ($i = 2; $i >= 0; $i--) {
    for ($j = 0; $j <= 2; $j++) {
        echo $a[$i][$j], " ";
    }
}

```

```

iii) for ($i = 0; $i <= 2; $i++) {
      for ($j = 2; $j >= 0; $j--) {
        echo $a[$i][$j], " ";
      }
    }

```

```

iv)  for ($i = 2; $i >= 0; $i--) {
      for ($j = 2; $j >= 0; $j--) {
        echo $a[$i][$j], " ";
      }
    }

```

```

v)   for ($j = 0; $j <= 2; $j++) {
      for ($i = 0; $i <= 2; $i++) {
        echo $a[$i][$j], " ";
      }
    }

```

```

vi)  for ($j = 0; $j <= 2; $j++) {
      for ($i = 2; $i >= 0; $i--) {
        echo $a[$i][$j], " ";
      }
    }

```

```

vii) for ($j = 2; $j >= 0; $j--) {
      for ($i = 0; $i <= 2; $i++) {
        echo $a[$i][$j], " ";
      }
    }

```

```

viii) for ($j = 2; $j >= 0; $j--) {
      for ($i = 2; $i >= 0; $i--) {
        echo $a[$i][$j], " ";
      }
    }

```

9. Write a PHP script that lets the user enter integer values in a 10×15 array and then displays the indexes of the positions that contain odd numbers.
10. Write a PHP script that lets the user enter numeric values in a 10×6 array and then displays the elements of the columns with even-numbered indexes (that is, column indexes 0, 2, and 4).
11. Write a PHP script that lets the user enter numeric values in a 12×8 array and then calculates and displays the sum of the elements that have even column indexes and odd row indexes.
12. Write a PHP script that lets the user enter numeric values in an 8×8 square array and then calculates the average value of the elements of its main diagonal and the average value of the elements of its antidiagonal. Try to calculate both average values within the same loop control structure.
13. Write a PHP script that creates and displays the following array.

	0	1	2	3	4
0	11	11	11	11	5
1	11	11	11	5	88
2	11	11	5	88	88
3	11	5	88	88	88
4	5	88	88	88	88

14. Write a PHP script that creates and displays the following array.

	0	1	2	3	4
0	0	11	11	11	5
1	11	0	11	5	88
\$a = 2	11	11	0	88	88
3	11	5	88	0	88
4	5	88	88	88	0

15. Write a PHP script that lets the user enter numeric values in a 5×4 array and then displays the row and column indexes of the positions that contain integers.

16. Write a PHP script that lets the user enter numeric values in a 10×4 array and then counts and displays the total number of negative elements.

17. Write a PHP script that lets the user enter words in a 3×4 array and then displays them with a space character between them.

18. Write a PHP script that lets the user enter words in a 20×14 array and then displays those who have less than five characters.

Hint: Use the `strlen()` function.

19. Write a PHP script that lets the user enter words in a 20×14 array and displays those that have less than five characters, then those that have less than 10 characters, and finally those that have less than 20 characters. Assume that the user enters only words with less than 20 characters.

Hint: Try to display the words using three counted loop structures nested one within the other.

Chapter 35

Tips and Tricks with Arrays

35.1 Introduction

Since arrays are handled with the same sequence, decision, and loop control structures that you learned about in previous chapters, there is no need to repeat all of that information here. However, what you will discover in this chapter is how to process each row or column of a two-dimensional array individually, how to solve problems that require the use of more than one array, and how to create a two-dimensional array from a one-dimensional array and vice versa.

35.2 Processing Each Row Individually

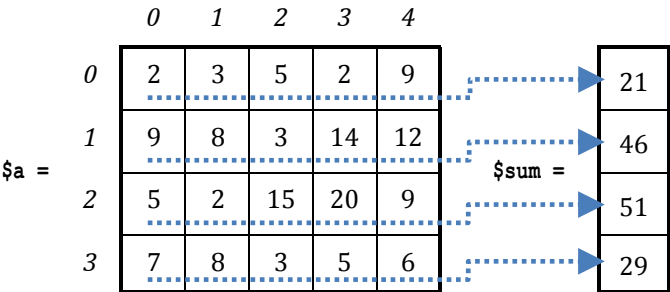
Processing each row individually means that every row is processed separately and the result of each row (which can be the sum, the average value, and so on) can be used individually for further processing. Suppose you have the following 4 × 5 array.

	0	1	2	3	4
0	2	3	5	2	9
1	9	8	3	14	12
2	5	2	15	20	9
3	7	8	3	5	6

Let’s try to find the sum of each row individually. There are actually two approaches that you can use. Both of these approaches iterate through rows.

First Approach – Creating an auxiliary array

In this approach, the script processes each row individually and creates an auxiliary array in which each position stores the sum of one row. This approach gives you much flexibility since you can use this new array later in your script for further processing. The auxiliary array is shown on the right.



Now, let’s write the corresponding code fragment. To more easily understand the process, the “from inner to outer” method is used. The following code fragment calculates the sum of the first row (row index 0) and stores the result in position 0 of the auxiliary array. Assume variable `$i` contains the value 0.

```
$s = 0;
for ($j = 0; $j <= COLUMNS - 1; $j++) {
    $s += $a[$i][$j];
}
$sum[$i] = $s;
```

This script can equivalently be written as

```
$sum[$i] = 0;
for ($j = 0; $j <= COLUMNS - 1; $j++) {
    $sum[$i] += $a[$i][$j];
}
```

Now, nesting this code fragment in a counted loop structure that iterates for all rows results in the following.

```
$sum = array();
for ($i = 0; $i <= ROWS - 1; $i++) {
    $sum[$i] = 0;
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $sum[$i] += $a[$i][$j];
    }
}
```

Second Approach – Just find it and process it!

This approach uses no auxiliary array; it just calculates and directly processes the sum. The code fragment is as follows.

```
for ($i = 0; $i <= ROWS - 1; $i++) {
    $sum = 0;
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $sum += $a[$i][$j];
    }
    process $sum;
}
```

What does *process \$sum* mean? It depends on the given problem. It may just display the sum, it may calculate the average value of each individual row and display it, or it may use the sum for calculating even more complex mathematical expressions.

For instance, the following example calculates and displays the average value of each row.

```
for ($i = 0; $i <= ROWS - 1; $i++) {
    $sum = 0;
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $sum += $a[$i][$j];
    }
}
```

```
$average = $sum / COLUMNS;
echo $average;
}
```

Exercise 35.2-1 Finding the Average Value

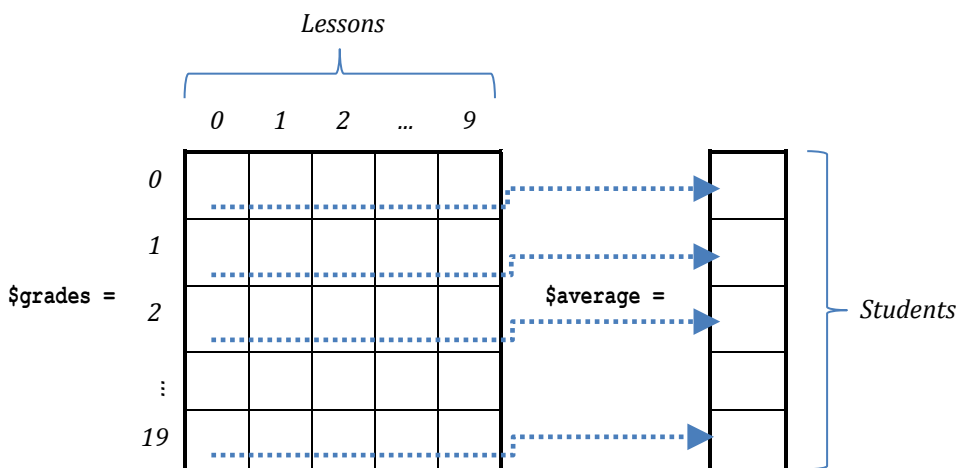
There are 20 students and each one of them has received his or her grades for 10 lessons. Write a PHP script that prompts the user to enter the grades of each student for all lessons and then calculates and displays, for each student, all average values that are greater than 89.

Solution

Since you've learned two approaches for processing each row individually, let's use them both.

First Approach – Creating an auxiliary array

In this approach, the script processes each row individually and creates an auxiliary array in which each position stores the average value of one row. The two required arrays are shown next.



After the array `$average` is created, the script can find and display all average values that are greater than 89. The PHP script is as follows.

file_35.2-1a.php

```
<?php
define("STUDENTS", 20);
define("LESSONS", 10);

$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "For student No. ", $i + 1, "... \n";
```

```

for ($j = 0; $j <= LESSONS - 1; $j++) {
    echo "enter grade for lesson No. ", $j + 1, ": ";
    $grades[$i][$j] = trim(fgets(STDIN));
}
}

//Create array $average
$average = array();
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    $average[$i] = 0;
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        $average[$i] += $grades[$i][$j];
    }
    $average[$i] /= LESSONS;
}

//Display all average values that are greater than 89
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    if ($average[$i] > 89) {
        echo $average[$i], "\n";
    }
}
?>

```

Second Approach – Just find it and display it!

This approach uses no auxiliary array; it just calculates and directly displays all average values that are greater than 89. The PHP script is as follows.

file_35.2-1b.php

```

<?php
define("STUDENTS", 20);
define("LESSONS", 10);

$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "For student No. ", $i + 1, "... \n";
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        echo "enter grade for lesson No. ", $j + 1, ": ";
        $grades[$i][$j] = trim(fgets(STDIN));
    }
}
}

```



```
//Calculate the average value of each row individually
//and directly display those who are greater than 89
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    $saverage = 0;
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        $saverage += $grades[$i][$j];
    }
    $saverage /= LESSONS;
    if ($saverage > 89) {
        echo $saverage, "\n";
    }
}
?>
```

35.3 Processing Each Column Individually

Processing each column individually means that every column is processed separately and the result of each column (which can be the sum, the average value, and so on) can be used individually for further processing. Suppose you have the following 4×5 array.

	0	1	2	3	4
0	2	3	5	2	9
1	9	8	3	14	12
2	5	2	15	20	9
3	7	8	3	5	6

As before, let's try to find the sum of each column individually. Yet again, there are two approaches that you can use. Both of these approaches iterate through columns.

First Approach – Creating an auxiliary array

In this approach, the script processes each column individually and creates an auxiliary array in which each position stores the sum of one column. This approach gives you much flexibility since you can use this new array later in your script for further processing.

	0	1	2	3	4
0	2	3	5	2	9
1	9	8	3	14	12
2	5	2	15	20	9
3	7	8	3	5	6

\$a =

	0	1	2	3	4
sum	23	21	26	41	36

\$sum =

Notice: The auxiliary array is shown at the bottom.

Now, let's write the corresponding code fragment. To more easily understand the process, the "from inner to outer" method is used again. The following code fragment calculates the sum of the first column (column index 0) and stores the result in position 0 of the auxiliary array. Assume variable `$j` contains the value 0.

```
$s = 0;
for ($i = 0; $i <= ROWS - 1; $i++) {
    $s += $a[$i][$j];
}
$sum[$j] = $s;
```

This script can equivalently be written as

```
$sum[$j] = 0;
for ($i = 0; $i <= ROWS - 1; $i++) {
    $sum[$j] += $a[$i][$j];
}
```

Now, nesting this code fragment in a counted loop structure that iterates for all columns results in the following.

```
$sum = array();
for ($j = 0; $j <= COLUMNS - 1; $j++) {
    $sum[$j] = 0;
    for ($i = 0; $i <= ROWS - 1; $i++) {
        $sum[$j] += $a[$i][$j];
    }
}
```

Second Approach – Just find it and process it!

This approach, as previously shown, uses no auxiliary array; it just calculates and directly processes the sum. The code fragment is as follows.

```
for ($j = 0; $j <= COLUMNS - 1; $j++) {  
    $sum = 0;  
    for ($i = 0; $i <= ROWS - 1; $i++) {  
        $sum += $a[$i][$j];  
    }  
    process $sum;  
}
```

Accordingly, the following code fragment calculates and displays the average value of each column.

```
for ($j = 0; $j <= COLUMNS - 1; $j++) {  
    $sum = 0;  
    for ($i = 0; $i <= ROWS - 1; $i++) {  
        $sum += $a[$i][$j];  
    }  
    echo $sum / ROWS;  
}
```

Exercise 35.3-1 *Finding the Average Value*

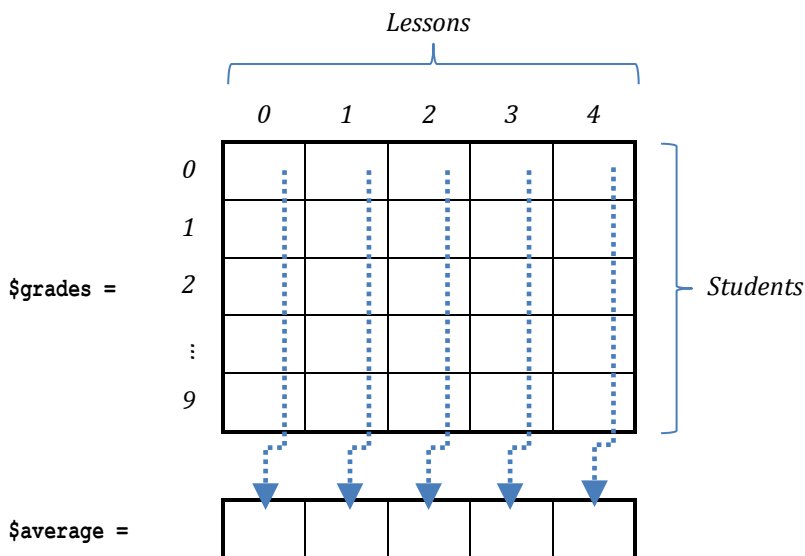
There are 10 students and each one of them has received his or her grades for five lessons. Write a PHP script that prompts the user to enter the grades of each student for all lessons and then calculates and displays, for each lesson, all average values that are greater than 89.

Solution

Since you've learned two approaches for processing each column individually, let's use them both.

First Approach – Creating an auxiliary array

In this approach, the script processes each column individually and creates an auxiliary array in which each position stores the average value of one column. The two required arrays are as follows.



After the array `$average` is created, the script can find and display all average values that are greater than 89. The PHP script is as follows.

file_35.3-1a.php

```
<?php
define("STUDENTS", 10);
define("LESSONS", 5);

$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "For student No. ", $i + 1, "... \n";
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        echo "enter grade for lesson No. ", $j + 1, " : ";
        $grades[$i][$j] = trim(fgets(STDIN));
    }
}

//Create array $average. Iterate through columns
$average = array();
for ($j = 0; $j <= LESSONS - 1; $j++) {
    $average[$j] = 0;
    for ($i = 0; $i <= STUDENTS - 1; $i++) {
        $average[$j] += $grades[$i][$j];
    }
}
```

```

    }
    $average[$j] /= STUDENTS;
}

//Display all average values than are greater than 89
for ($j = 0; $j <= LESSONS - 1; $j++) {
    if ($average[$j] > 89) {
        echo $average[$j], "\n";
    }
}
?>

```

Second Approach – Just find it and display it!

This approach uses no auxiliary array; it just calculates and directly displays all average values that are greater than 89. The PHP script is as follows.

file_35.3-1b.php

```

<?php
define("STUDENTS", 10);
define("LESSONS", 5);

$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "For student No. ", $i + 1, "... \n";
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        echo "enter grade for lesson No. ", $j + 1, ", ": ";
        $grades[$i][$j] = trim(fgets(STDIN));
    }
}

//Calculate the average value of each column individually
//and directly display those who are greater than 89
for ($j = 0; $j <= LESSONS - 1; $j++) {
    $average = 0;
    for ($i = 0; $i <= STUDENTS - 1; $i++) {
        $average += $grades[$i][$j];
    }
    $average /= STUDENTS;
    if ($average > 89) {
        echo $average, "\n";
    }
}

```

```
}  
?>
```

35.4 How to Use One-Dimensional Along with Two-Dimensional Arrays

So far, every example or exercise has used either a one-dimensional or a two-dimensional array. But what if a problem requires you to use them both? Next you will find some exercises that show you how various arrays can be used together to solve a problem.

Exercise 35.4-1 Finding the Average Value

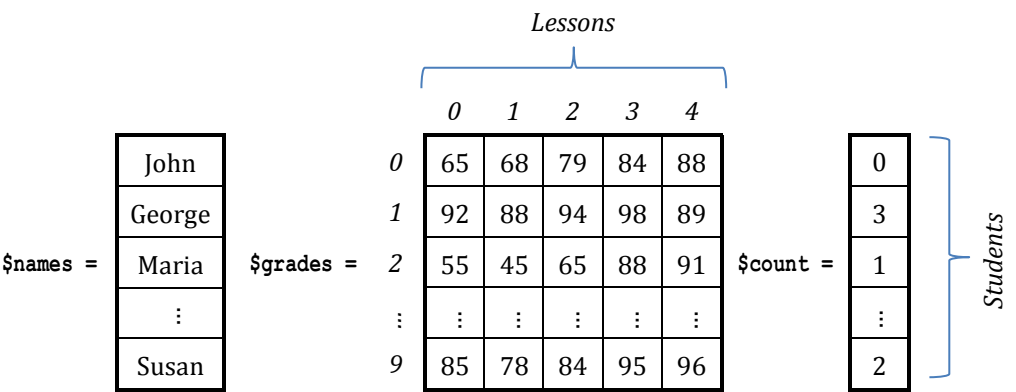
There are 10 students and each one of them has received his or her grades for five lessons. Write a PHP script that prompts the user to enter the name of each student and the grades for all lessons and then calculates and displays the names of the students who have more than one grade greater than 89.

Solution

In this exercise, you need a one-dimensional array to store the names of the students and a two-dimensional array to store the grades for each student for each lesson. There are actually two approaches. Which one to use depends clearly on you! If you decide that, in the two-dimensional array, the rows should refer to students and the columns should refer to lessons then you can use the first approach discussed below. If you decide that the rows should refer to lessons and the columns should refer to students then you can use the second approach.

First Approach – Rows for students, columns for lessons

In this approach, the two-dimensional array should have 10 rows, one for every student and 5 columns, one for every lesson. All other arrays can be placed in relation to this two-dimensional array as follows.



Notice: The auxiliary array `$count` is created by the script and contains the number of grades for each student that are greater than 89.

Now, let's see how to read values and store them in the arrays `$names` and `$grades`. One simple solution would be to use one loop control structure for reading names, and another, independent, loop control structure for reading grades. However, it is not very practical for the user to first enter all names and later enter all grades. A more practical way would be to prompt the user to enter one student name and then all of his or her grades, then another student name and again all of his or her grades, and so on. The solution is as follows.

file_35.4-1a.php

```
<?php
define("STUDENTS", 10);
define("LESSONS", 5);

//Read names and grades all together
$names = array();
$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "Enter name for student No. ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        echo "Enter grade No. ", $j + 1, " for ", $names[$i], ": ";
        $grades[$i][$j] = trim(fgets(STDIN));
    }
}

//Create array $count
$count = array();
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    $count[$i] = 0;
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        if ($grades[$i][$j] > 89) {
            $count[$i]++;
        }
    }
}

//Displays the names of the students who have
//more than one grades greater than 89
for ($i = 0; $i <= STUDENTS - 1; $i++) {
```

```

if ($count[$i] > 1) {
    echo $names[$i], "\n";
}
}
?>

```

Second Approach – Rows for lessons, columns for students

In this approach, the two dimensional array should have 5 rows, one for every lesson and 10 columns, one for every student. All other arrays can be placed in relation to this two-dimensional array, as shown next.

		Students				
\$names =		John	George	Maria	...	Susan
		0	1	2	...	9
\$grades =	0	65	92	55	...	85
	1	68	88	45	...	78
	2	79	94	65	...	84
	3	84	98	88	...	95
	4	88	89	91	...	96
		Lesson				
\$count =		0	3	1	...	2

Notice: The auxiliary array `$count` is created by the script and contains the number of grades for each lesson that are greater than 89.

The solution is as follows.

file_35.4-1b.php

```

<?php
define("STUDENTS", 10);
define("LESSONS", 5);

//Read names and grades together
$names = array();
$grades = array(array());
for ($j = 0; $j <= STUDENTS - 1; $j++) {
    echo "Enter name for student No. ", $j + 1, ": ";
    $names[$j] = trim(fgets(STDIN));
}

```



```
for ($i = 0; $i <= LESSONS - 1; $i++) {
    echo "Enter grade No. ", $i + 1, " for ", $names[$j], ": ";
    $grades[$i][$j] = trim(fgets(STDIN));
}
}

//Create array $count
$count = array();
for ($j = 0; $j <= STUDENTS - 1; $j++) {
    $count[$j] = 0;
    for ($i = 0; $i <= LESSONS - 1; $i++) {
        if ($grades[$i][$j] > 89) {
            $count[$j]++;
        }
    }
}

//Displays the names of the students who have
//more than one grades greater than 89
for ($j = 0; $j <= STUDENTS - 1; $j++) {
    if ($count[$j] > 1) {
        echo $names[$j], "\n";
    }
}

?>
```

35.5 Creating a One-Dimensional Array from a Two-Dimensional Array

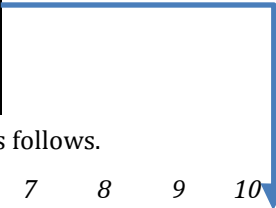
To more easily understand how to create a one-dimensional array from a two-dimensional array, let's use an example.

Write a script that creates a one-dimensional array of 12 elements from a two-dimensional array of 3×4 as follows: The elements of the first column of the two-dimensional array should be placed in the first three positions of the one-dimensional array, the elements of the second column should be placed in the next three positions, and so on.

An example of a 3×4 array is as follows.

	0	1	2	3
0	5	9	3	2
1	11	12	4	1
2	10	25	22	18

\$a =



Accordingly, the new one-dimensional array will be as follows.

	0	1	2	3	4	5	6	7	8	9	10	11
\$b =	5	11	10	9	12	25	3	4	22	2	1	18

When the user enters data, the script can iterate, as usual, through rows but to create the new one-dimensional array, iterating through columns is more convenient. The PHP script is presented next.

file_35.5.php

```
<?php
define("ROWS", 3);
define("COLUMNS", 4);
define("ELEMENTS", ROWS * COLUMNS);

$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        echo "Enter a value for element ", $i , ", ", $j, ": ";
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

$b = array();
$k = 0; //This is the index of the new array.
for ($j = 0; $j <= COLUMNS - 1; $j++) { //Iterate through columns
    for ($i = 0; $i <= ROWS - 1; $i++) {
        $b[$k] = $a[$i][$j];
        $k++;
    }
}

for ($k = 0; $k <= ELEMENTS - 1; $k++) {
    echo $b[$k], " ";
}
```

```
}
?>
```

35.6 Creating a Two-Dimensional Array from a One-Dimensional Array

To more easily understand how to create a two-dimensional array from a one-dimensional array, let's use an example.

Write a PHP script that creates a two-dimensional array of 3×4 from a one-dimensional array of 12 elements as follow: The first three elements of the one-dimensional array should be placed in the first column of the two-dimensional array, the next three elements of the one-dimensional array should be placed in the next column of the two-dimensional array and so on.

An example of a 3×4 array is as follows.

	0	1	2	3	4	5	6	7	8	9	10	11
\$a =	5	11	10	9	12	25	3	4	22	2	1	18

Accordingly, the new two-dimensional array will be as follows.

		0	1	2	3
	0	5	9	3	2
\$b =	1	11	12	4	1
	2	10	25	22	18



When the user enters data, the script can iterate, as usual, through rows but to create the new one-dimensional array, iterating through columns is more convenient. The PHP script is presented next.

file_35.6.php

```
<?php
define("ROWS", 3);
define("COLUMNS", 4);
define("ELEMENTS", ROWS * COLUMNS);

$a = array();
for ($k = 0; $k <= ELEMENTS - 1; $k++) {
    echo "Enter a value for element ", $k, ": ";
    $a[$k] = trim(fgets(STDIN));
}

$b = array(array());
```

```

$k = 0; //This is the index of array $a.
for ($j = 0; $j <= COLUMNS - 1; $j++) { //Iterate through columns
    for ($i = 0; $i <= ROWS - 1; $i++) {
        $b[$i][$j] = $a[$k];
        $k++;
    }
}

for ($i = 0; $i <= ROWS - 1; $i++) { //Iterate through rows
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        echo $b[$i][$j], "\t";
    }
    echo "\n";
}
?>

```

35.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Processing each row individually means that every row is processed separately, and the result of each row can then be used individually for further processing.
2. The following code fragment displays the word “Okay” when the sum of the elements of each column is less than 100.

```

for ($i = 0; $i <= ROWS - 1; $i++) {
    $sum = 0;
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $sum += $a[$i][$j];
    }

    if ($sum < 100) echo "Okay";
}

```

3. Processing each column individually means that every column is processed separately and the result of each column can be then used individually for further processing.
4. The following code fragment displays the sum of the elements of each column.

```

$sum = 0;
for ($j = 0; $j <= COLUMNS - 1; $j++) {
    for ($i = 0; $i <= ROWS - 1; $i++) {
        $sum += $a[$i][$j];
    }
}

```

```
}  
  
echo $sum;  
}
```

5. Suppose that there are 10 students and each one of them has received his or her grades for five lessons. Given this information, it is possible to design an array so that the rows refer to students and the columns refer to lessons, but not the other way around.
6. A one-dimensional array can be created from a two-dimensional array, but not the opposite.
7. A one-dimensional array can be created from a three-dimensional array.

35.8 Review Questions: Multiple Choice

Select the correct answer for each of the following statements.

1. The following code fragment

```
$sum = array();  
for ($i = 0; $i <= ROWS - 1; $i++) {  
    $sum[$i] = 0;  
    for ($j = 0; $j <= COLUMNS - 1; $j++) {  
        $sum[$i] += $a[$i][$j];  
    }  
    echo $sum[$i];  
}
```

- a. displays the sum of the elements of each row.
- b. displays the sum of the elements of each column.
- c. displays the sum of all the elements of the array.
- d. none of the above

2. The following code fragment

```
for ($j = 0; $j <= COLUMNS - 1; $j++) {  
    $sum = 0;  
    for ($i = 0; $i <= ROWS - 1; $i++) {  
        $sum += $a[$i][$j];  
    }  
    echo $sum;  
}
```

- a. displays the sum of the elements of each row.
- b. displays the sum of the elements of each column.
- c. displays the sum of all the elements of the array.
- d. none of the above

3. The following code fragment

```
$k = 0;
for ($i = ROWS - 1; $i >= 0; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j--) {
        $b[$k] = $a[$i][$j];
        $k++;
    }
}
```

- creates a one-dimensional array from a two-dimensional array.
- creates a two-dimensional array from a one-dimensional array.
- none of the above

4. The following code fragment

```
$k = 0;
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = COLUMNS - 1; $j >= 0; $j--) {
        $b[$i][$j] = $a[$k];
        $k++;
    }
}
```

- creates a one-dimensional array from a two-dimensional array.
- creates a two-dimensional array from a one-dimensional array.
- none of the above

35.9 Review Exercises

Complete the following exercises.

- There are 15 students and each one of them has received his or her grades for five tests. Write a PHP script that lets the user enter the grades (as a percentage) for each student for all tests. It then calculates and displays, for each student, the average grade as a letter grade according to the following table.

Grade	Percentage
A	90 – 100
B	80 – 89
C	70 – 79
D	60 – 69
E / F	0 – 59

2. On Earth, a free-falling object has an acceleration of 9.81 m/s^2 downward. This value is denoted by g . A student wants to calculate that value using an experiment. She allows five different objects to fall downward from a known height and measures the time they need to reach the floor. She does this 10 times for each object. Then, using a formula she calculates g for each object, for each fall. But since her chronometer is not so accurate, she needs a PHP script that lets her enter all calculated values of g in a 5×10 array. The script should then calculate and display
 - a. for each object, the average value of g (per fall)
 - b. for each fall, the average value of g (per object)
 - c. the overall average value of g
3. A basketball team with 15 players plays 12 matches. Write a PHP script that lets the user enter, for each player, the number of points scored in each match. The script should then display
 - a. for each player, the total number of points scored
 - b. for each match, the total number of points scored
4. Write a PHP script that lets the user enter the hourly measured temperatures of 20 cities for a period of one day, and then displays the hours in which the average temperature of all the cities was below 10 degrees Fahrenheit.
5. In a football tournament, a football team with 24 players plays 10 matches. Write a PHP script that lets the user enter, for each player, a name as well as the number of goals scored in each match. The script should then display
 - a. for each player, a name and the average number of goals scored
 - b. for each match, the index number of the match (1, 2, 3, and so on) and the total number of goals scored
6. There are 12 students and each one of them has received his or her grades for six lessons. Write a PHP script that lets the user enter the name of the student as well as his or her grades in all lessons and then displays
 - a. for each student, his or her name and average grade
 - b. for each lesson, the average grade
 - c. the names of the students who have an average grade less than 60
 - d. the names of the students who have an average grade greater than 89, and the message "Bravo!" next to it

Assume that the user enters values between 0 and 100.

7. In a song contest, each artist sings a song of his or her choice. There are five judges and 15 artists, each of whom is scored for his or her performance. Write a PHP script that prompts the user to enter the names of the judges, the names of the artists, the title of the song that each artist sings, and the score they get from each judge. The script should then display
 - a. for each artist, his or her name, the title of the song, and his or her total score

- b. for each judge, his or her name and the average value of the score he or she gave (per artist)
8. The Body Mass Index (BMI) is often used to determine whether a person is overweight or underweight for his or her height. The formula used to calculate BMI is

$$BMI = \frac{weight \cdot 703}{height^2}$$

Write a PHP script that lets the user enter into two arrays the weight (in pounds) and height (in inches) of 30 people, measured on a monthly basis, for a period of one year (January to December). The script should then calculate and display

- a. for each person, his or her average weight, average height, and average BMI (per month)
 - b. for each person, his or her BMI in May and in August

Please note that all people are adults but some of them are between the ages of 18 and 25. This means they may still grow taller, thus their height might be different each month!

- a. for each consumer, the amount of kWh consumed and the amount of money that must be paid given a cost of each kWh of \$0.07 and a value added tax (VAT) of 19%
 - b. the overall average consumption and the overall average amount of money that must be paid (per consumer)
10. Write a PHP script that prompts the user to enter an amount in US dollars and calculates and displays the corresponding currency value in Euros, British Pounds Sterling, Australian Dollars, and Canadian Dollars. The tables below contain the exchange rates for each currency for a one-week period. The script should calculate the average value of each currency and do the conversions based on that average value.

\$currency =	British Pound Sterling
	Euro
	Canadian Dollar
	Australian Dollar

\$rate =	1.579	1.577	1.572	1.580	1.584
	1.269	1.270	1.265	1.240	1.255
	0.895	0.899	0.884	0.888	0.863
	0.811	0.815	0.822	0.829	0.819

11. Gross pay depends on the pay rate and the total number of hours worked per week. However, if someone works more than 40 hours, he or she gets paid time-and-a-half for all hours worked over 40. Write a PHP script that lets the user enter a pay rate as well as the names of 10 employees and the

number of hours that they worked each day (Monday to Friday). The script should then calculate and display

- a. the names of the employees who worked overtime
 - b. for each employee, his or her name and the average gross pay (per day)
 - c. for each employee, his or her name, the name of the day he or she worked overtime (more than 8 hours), and the message "Overtime!"
 - d. for each day, the name of the day and the total gross pay
12. Write a PHP script to create a one-dimensional array of 12 elements from the two-dimensional array shown below, as follows: the first row of the two-dimensional array should be placed in the first four positions of the one-dimensional array, the second row of the two-dimensional array should be placed in the next four positions of the one-dimensional array, and the last row of the two-dimensional array should be placed in the last four positions of the one-dimensional array.

\$a =	9	9	2	6
	4	1	10	11
	12	15	7	3

13. Write a PHP script to create a 3×3 array from the one-dimensional array shown below, as follows: the first three elements of the one-dimensional array should be placed in the last row of the two-dimensional array, the next three elements of the one-dimensional array should be placed in the second row of the two-dimensional array, and the last three elements of the one-dimensional array should be placed in the first row of the two-dimensional array.

\$a =	16	12	3	5	6	9	18	19	20

Chapter 36

Flowcharts with Arrays

36.1 Introduction

In this chapter you will learn nothing new about flowcharts. Earlier, you learned that an array is actually just a special type of variable, in that an array can hold multiple values at the same time. In flowcharts, you can use arrays in the same way that you use variables.

36.2 Converting PHP Scripts to Flowcharts

Since there is nothing new to discuss, let's directly proceed to solve some exercises!

Exercise 36.2-1 *Designing the Flowchart*

Design the flowchart that corresponds to the following PHP script.

```
<?php
define("ELEMENTS", 100);

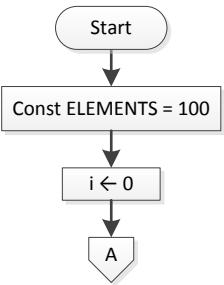
$values = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $values[$i] = trim(fgets(STDIN));
}

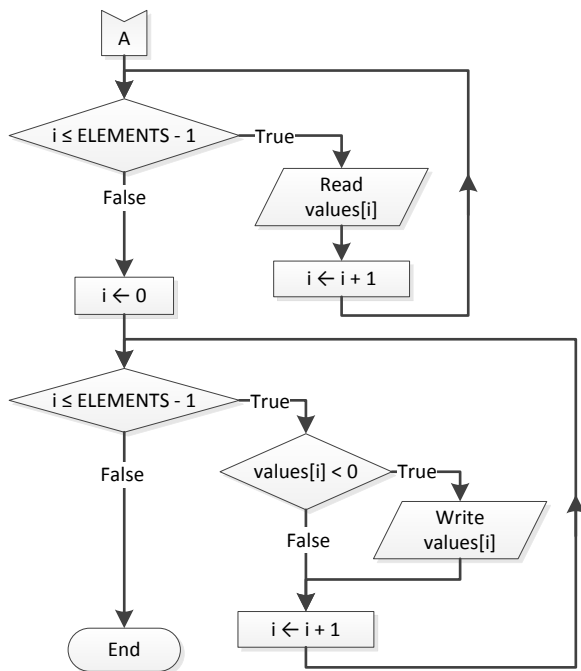
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($values[$i] < 0) {
        echo $values[$i];
    }
}

?>
```

Solution

In this PHP script, there are two counted loop structures, the second of which nests a single-alternative decision structure. The corresponding flowchart is presented next.





Exercise 36.2-2 *Designing the Flowchart*

Design the flowchart that corresponds to the following PHP script.

```

<?php
define("ELEMENTS", 50);

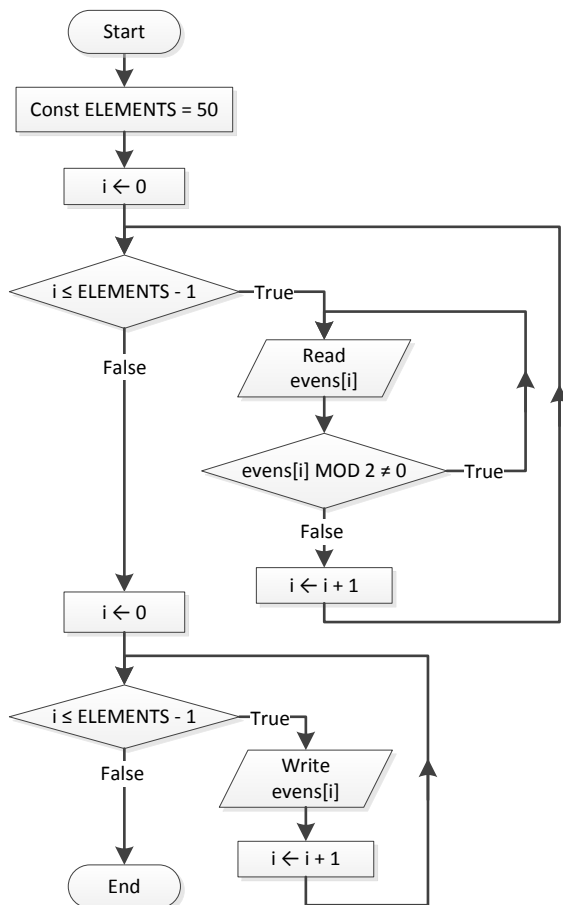
$evens = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    do {
        $evens[$i] = trim(fgets(STDIN));
    } while ($evens[$i] % 2 != 0);
}

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    echo $evens[$i];
}
?>

```

Solution

In this PHP script, there are two counted loop structures, the first of which nests a post-test loop structure. The corresponding flowchart is presented next.



Exercise 36.2-3 Designing the Flowchart

Design the flowchart that corresponds to the following PHP script.

```

<?php
define("ROWS", 20);
define("COLUMNS", 10);

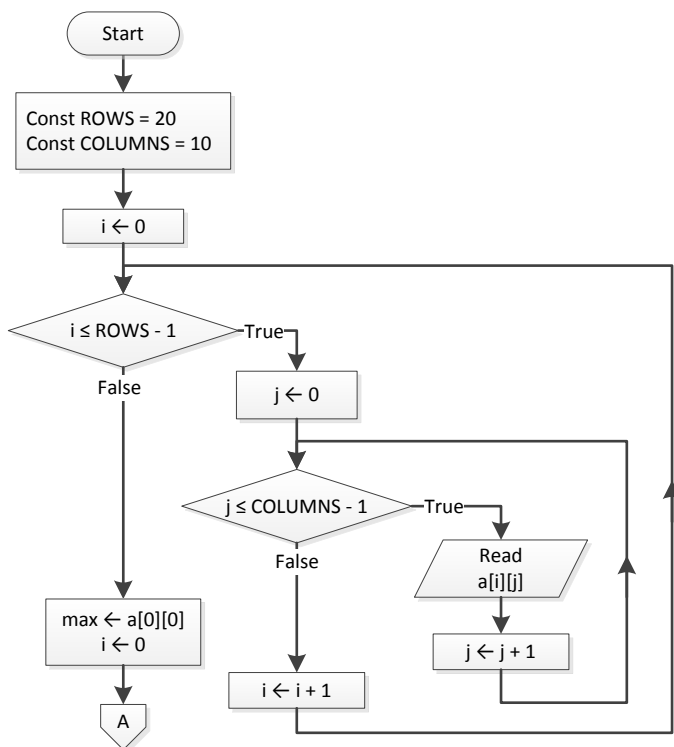
$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

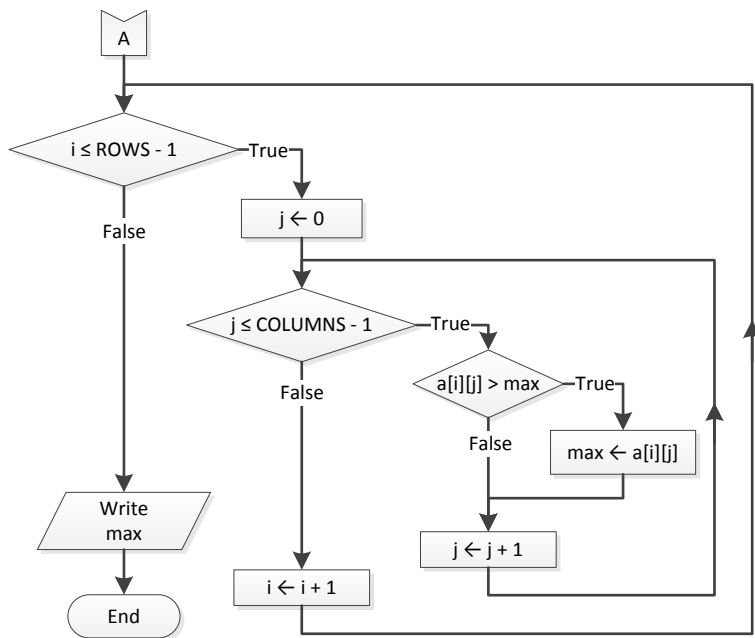
$max = $a[0][0];
    
```

```
for ($i = 0; $i <= ROWS - 1; $i++) {  
    for ($j = 0; $j <= COLUMNS - 1; $j++) {  
        if ($a[$i][$j] > $max) {  
            $max = $a[$i][$j];  
        }  
    }  
}  
  
echo $max;  
?>
```

Solution

In this PHP script, there are two nested loop control structures, the second of which nests a single-alternative decision structure. The corresponding flowchart is presented next.



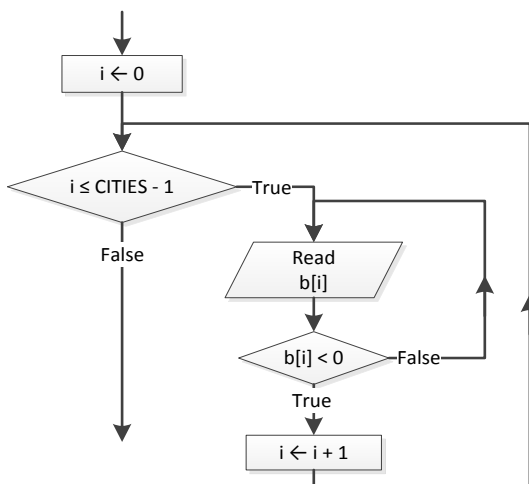


36.3 Converting Flowcharts to PHP Scripts

As already stated in previous chapters, this conversion is not always an easy one. There are cases in which the flowchart designers follow no particular rules, so the initial flowchart may need some modifications before it can become a PHP script.

Exercise 36.3-1 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart fragment.



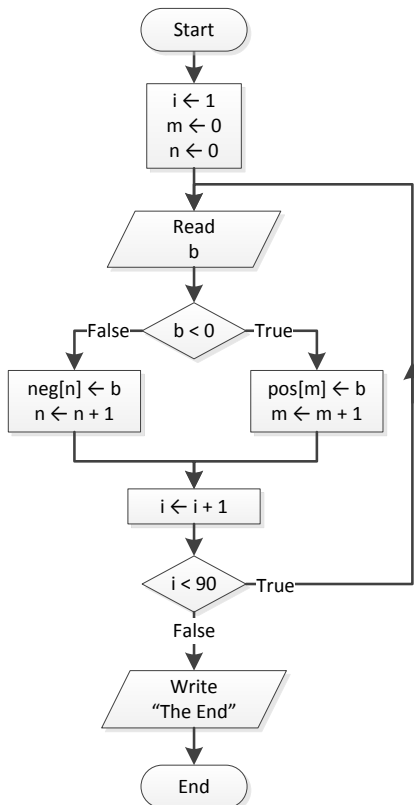
Solution

This exercise is quite simple. A post-test loop structure is nested within a pre-test loop structure (or in a counted loop structure, if you prefer). The only obstacle you must overcome is that the true and false paths of the inner post-test loop structure are not quite in the right positions since the true path, and not the false path, must go upwards. As you have already learned, it is possible to switch the two paths, but you also need to negate the corresponding Boolean expression. The code fragment is as follows.

```
for ($i = 0; $i <= CITIES - 1; $i++) {
    do {
        $b[$i] = trim(fgets(STDIN));
    } while ($b[$i] >= 0);
}
```

Exercise 36.3-2 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart.



Solution

In this exercise, a dual-alternative decision structure is nested within a post-test loop structure. The corresponding PHP script is as follows.

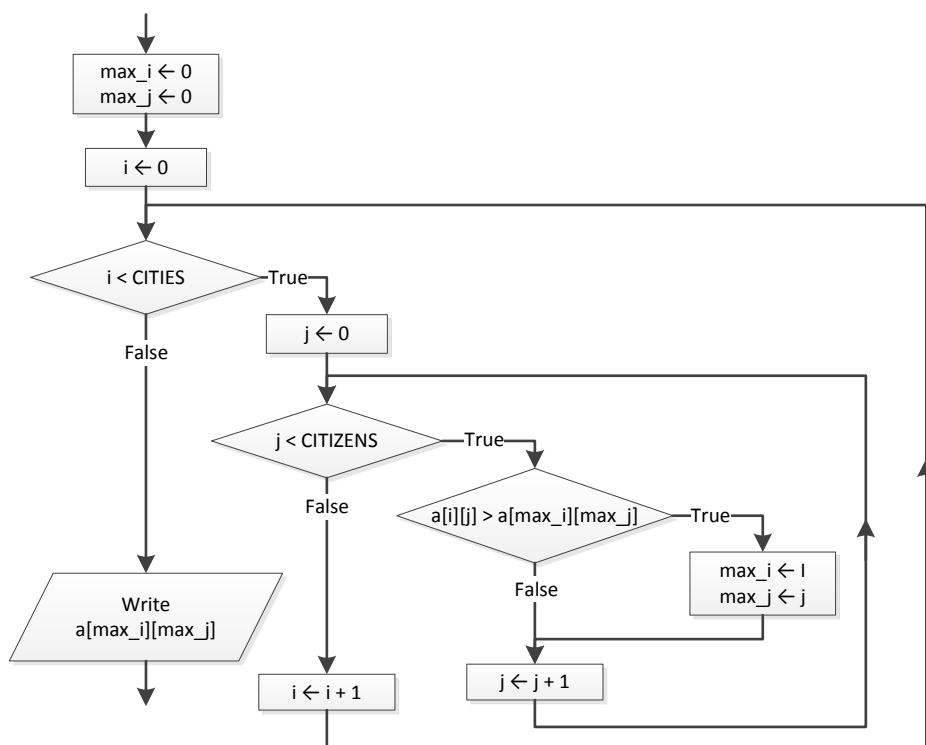
```
<?php
    $i = 1;
    $m = 0;
    $n = 0;
    $pos = array();
    $neg = array();
    do {
        $b = trim(fgets(STDIN));
        if ($b < 0) {
            $pos[$m] = $b;
            $m++;
        }
        else {
            $neg[$n] = $b;
            $n++;
        }
        $i++;
    } while ($i < 90);

    echo "The End";

?>
```

Exercise 36.3-3 *Writing the PHP Script*

Write the PHP script that corresponds to the following flowchart fragment.



Solution

This flowchart has one single-decision structure nested within a pre-test loop structure, which in turn is nested within another pre-test loop structure.

Since variables *i* and *j* are used as index positions for array *a*, you can assume that they are integers. Therefore, it is permitted to alter the two Boolean expressions to *i* <= CITIES - 1 and *j* <= CITIZENS - 1, respectively. This change helps you write the corresponding PHP script using counted loop structures, as follows.

```

$max_i = 0;
$max_j = 0;
for ($i = 0; $i <= CITIES - 1; $i++) {
    for ($j = 0; $j <= CITIZENS - 1; $j++) {
        if ($a[$i][$j] > $a[$max_i][$max_j]) {
            $max_i = $i;
            $max_j = $j;
        }
    }
}

```

```
echo $a[$max_i][$max_j];
```

36.4 Review Exercises

Complete the following exercises.

1. Design the flowchart that corresponds to the following PHP script.

```
<?php
define("ELEMENTS", 30);

$values = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    echo "Enter a two-digit integer: ";
    $values[$i] = trim(fgets(STDIN));
}

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $digit1 = intval($values[$i] / 10);
    $digit2 = $values[$i] % 10;
    if ($digit1 < $digit2) {
        echo $values[$i];
    }
}
?>
```

2. Design the flowchart that corresponds to the following PHP script.

```
<?php
define("ELEMENTS", 20);

$values = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    echo "Enter a three-digit integer: ";
    $values[$i] = trim(fgets(STDIN));
}

$sum = 0;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $digit3 = $values[$i] % 10;
    $r = intval($values[$i] / 10);

    $digit2 = $r % 10;
    $digit1 = intval($r / 10);
```

```
        if ($values[$i] == $digit3 * 100 + $digit2 * 10 + $digit1) {
            $sum += $values[$i];
        }
    }
    echo $sum;
?>
```

3. Design the flowchart that corresponds to the following PHP script.

```
<?php
define("N", 10);

$a = array(array());
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

//Calculate Sum
$sum = 0;
for ($k = 0; $k <= N - 1; $k++) {
    $sum += $a[$k][$k];
}

echo "Sum = ", $sum;
?>
```

4. Design the flowchart that corresponds to the following PHP script.

```
<?php
define("N", 10);

$a = array(array());
for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        if ($i == $j)
            $a[$i][$j] = "*";
        elseif ($i > $j)
            $a[$i][$j] = "-";
        else
            $a[$i][$j] = "+";
    }
}
```

```

    }
}

for ($i = 0; $i <= N - 1; $i++) {
    for ($j = 0; $j <= N - 1; $j++) {
        echo $a[$i][$j];
    }
}

?>

```

5. Design the flowchart that corresponds to the following code fragment.

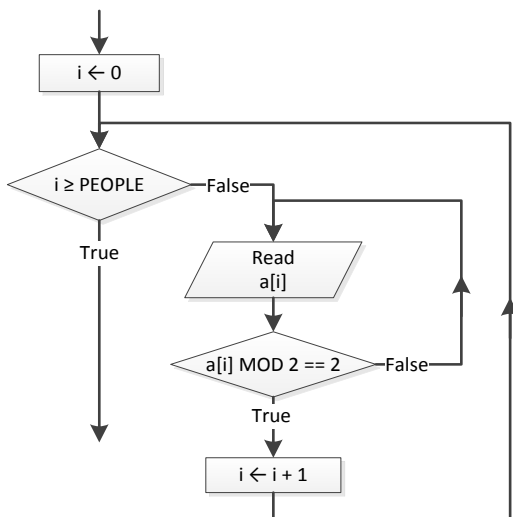
```

$average = array();
for ($i = 0; $i <= ROWS - 1; $i++) {
    $average[$i] = 0;
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $average[$i] += $values[$i][$j];
    }
    $average[$i] /= COLUMNS;
}

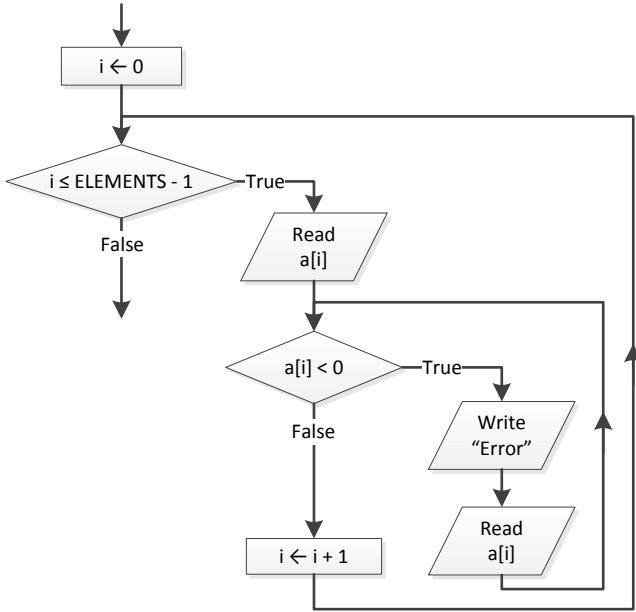
for ($i = 0; $i <= ROWS - 1; $i++) {
    if ($average[$i] > 89)
        echo $average[$i];
}

```

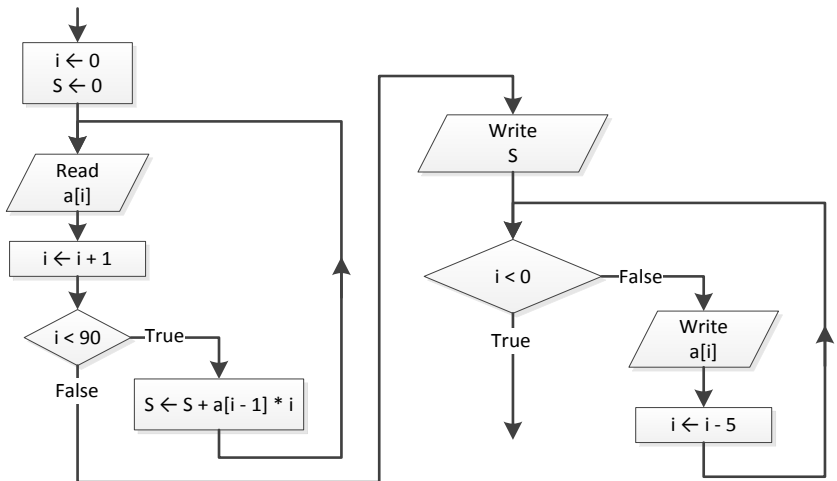
6. Write the PHP script that corresponds to the following flowchart fragment.



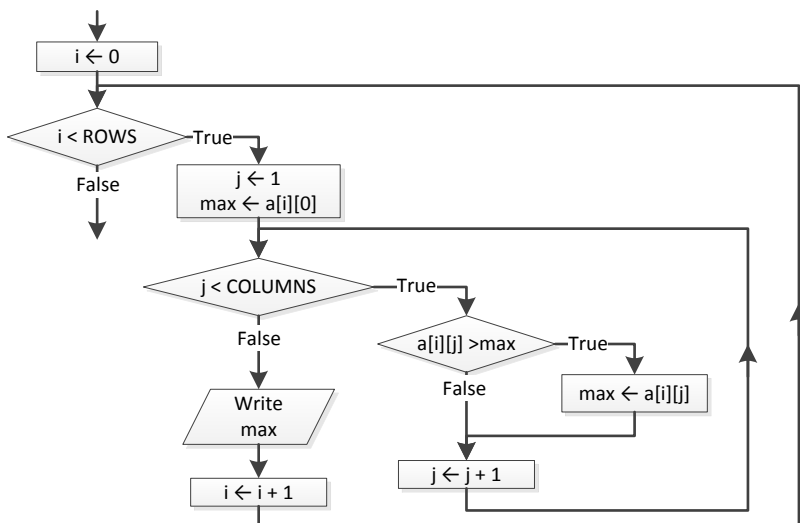
7. Write the PHP script that corresponds to the following flowchart fragment.



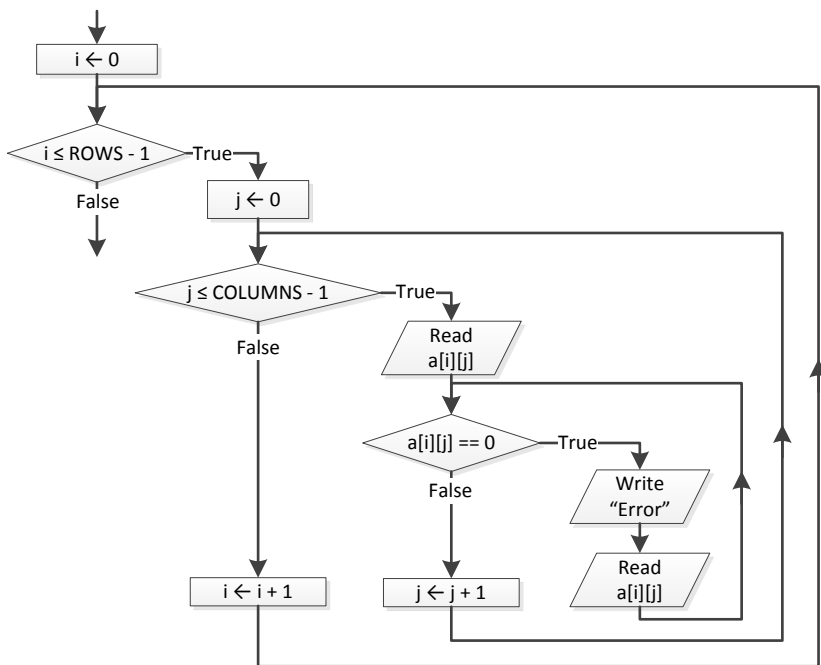
8. Write the PHP script that corresponds to the following flowchart fragment.



9. Write the PHP script that corresponds to the following flowchart fragment.



10. Write the PHP script that corresponds to the following flowchart fragment.



Chapter 37

More Exercises with Arrays

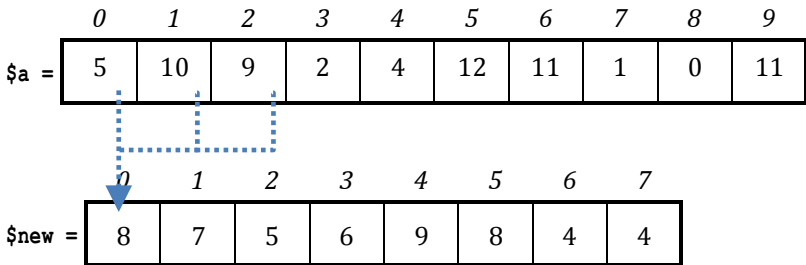
37.1 Simple Exercises with Arrays

Exercise 37.1-1 *Creating an Array that Contains the Average Values of its Neighboring Elements*

Design a flowchart and write the corresponding PHP script that lets the user enter 100 positive numerical values into an array. Then the flowchart, and consequently the script, should create a new array of 98 elements. This new array should contain, in each position the average value of the three elements that exist in the current and the next two positions of the given array.

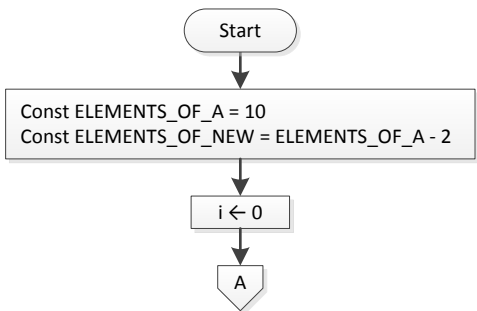
Solution

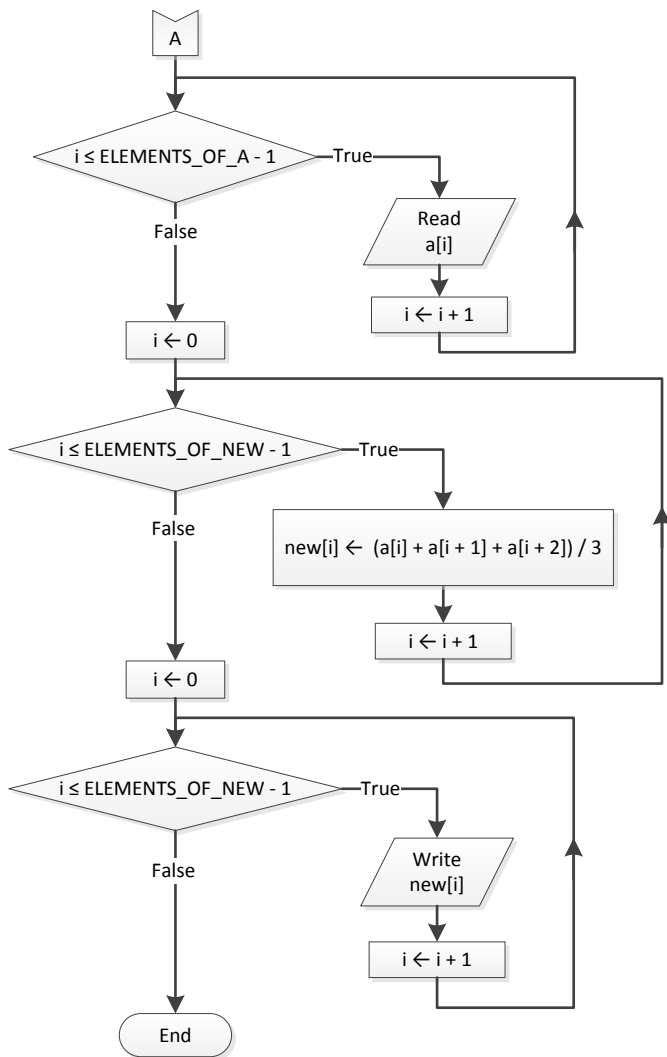
Let's try to understand this exercise through an example using 10 elements.



Array `$new` is the new array that is created. In array `$new`, the element at position 0 is the average value of the elements in the current and the next two positions of array `$a`; that is, $(5 + 10 + 9) / 3 = 8$. The element at position 1 is the average value of the elements in the current and the next two positions of array `$a`; that is, $(10 + 9 + 2) / 3 = 7$, and so on.

The flowchart is as follows.





The corresponding PHP script is as follows.

file_37.1-1.php

```

<?php
define("ELEMENTS_OF_A", 10);
define("ELEMENTS_OF_NEW", ELEMENTS_OF_A - 2);

$a = array();
for ($i = 0; $i <= ELEMENTS_OF_A - 1; $i++) {
    $a[$i] = trim(fgets(STDIN));
}
  
```

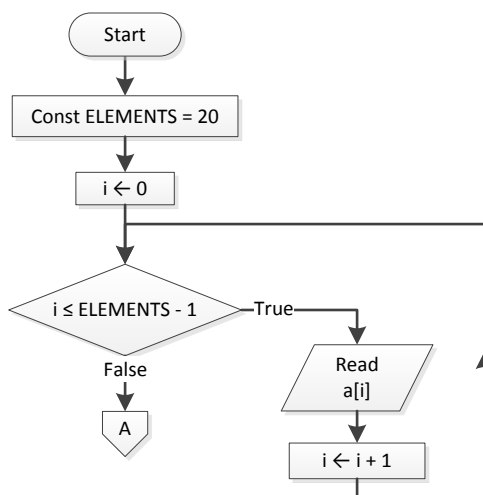
```
$new = array();  
for ($i = 0; $i <= ELEMENTS_OF_NEW - 1; $i++) {  
    $new[$i] = ($a[$i] + $a[$i + 1] + $a[$i + 2]) / 3;  
}  
  
for ($i = 0; $i <= ELEMENTS_OF_NEW - 1; $i++) {  
    echo $new[$i], "\t";  
}  
?  
?>
```

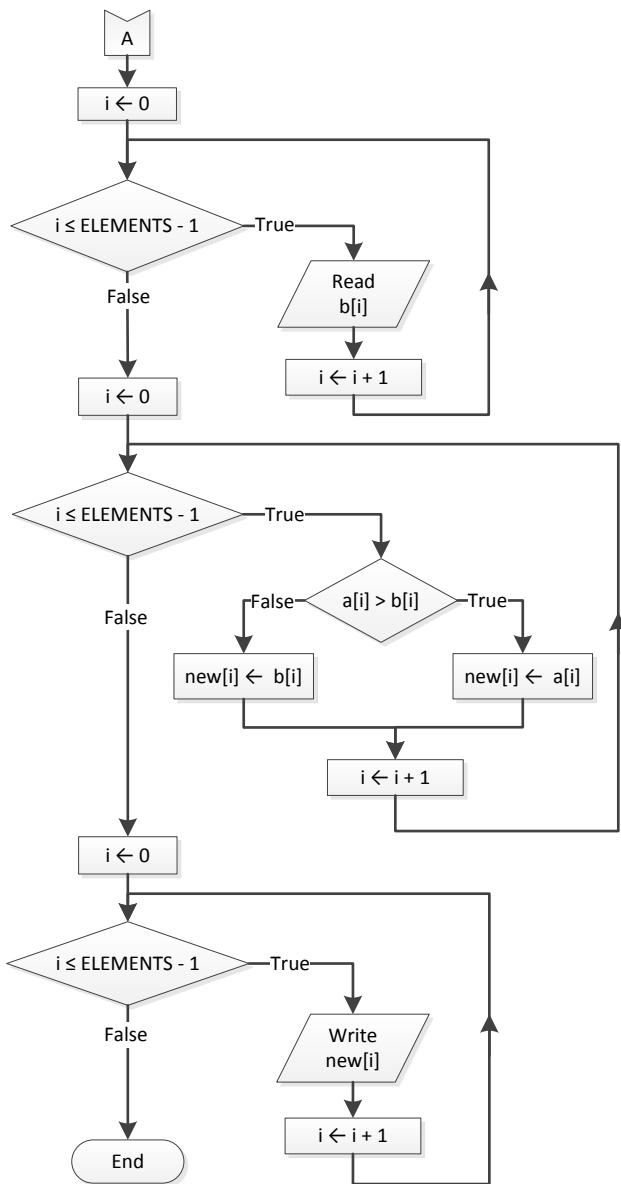
Exercise 37.1-2 *Creating an Array with the Greatest Values*

Design a flowchart and write the corresponding PHP script that lets the user enter numerical values into arrays a and b of 20 elements each. Then the algorithm, and consequently the script, should create a new array new of 20 elements. The new array should contain in each position the greatest value of arrays a and b of the corresponding position.

Solution

Nothing new here! You need four counted loop structures: two counted loop structures to read the values for arrays a and b , one for creating the array new , and one to display array new on the screen. The flowchart is presented next.





The corresponding PHP script is shown here.

file_37.1-2.php

```

<?php
define("ELEMENTS", 20);

//Read arrays $a and $b
$a = array();
    
```

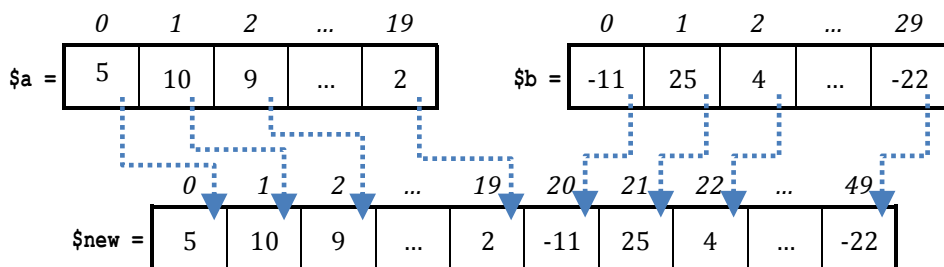
```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    $a[$i] = trim(fgets(STDIN));  
}  
$b = array();  
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    $b[$i] = trim(fgets(STDIN));  
}  
  
//Create array $new  
$new = array();  
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    if ($a[$i] > $b[$i]) {  
        $new[$i] = $a[$i];  
    }  
    else {  
        $new[$i] = $b[$i];  
    }  
}  
  
//Display array $new  
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    echo $new[$i], "\n";  
}  
?>
```

Exercise 37.1-3 *Merging One-Dimensional Arrays*

Write a PHP script that lets the user enter numerical values into arrays `$a` and `$b` of 20 and 30 elements correspondingly. Then, the script should create a new array `$new` of 50 elements. This new array should contain in the first 20 positions the elements of array `$a`, and in the next 30 positions the elements of array `$b`.

Solution

As you can see in the example presented next, there is a one-to-one match between the index positions of the elements of array `$a` and those of array `$new`. The element from position 0 of array `$a` is stored in position 0 of array `$new`, the element from position 1 of array `$a` is stored in position 1 of array `$new`, and so on. However, there is no one-to-one match between the index positions of the elements of array `$b` and those of array `$new`. The element from position 0 of array `$a` should be stored in position 20 of array `$new`, the element from position 1 of array `$a` should be stored in position 21 of array `$new`, and so on.



In order to assign the values of array `$a` to array `$new` you can use the following code fragment.

```
for ($i = 0; $i <= ELEMENTS_OF_A - 1; $i++) {
    $new[$i] = $a[$i];
}
```

However, to assign the values of array `$b` to array `$new` your code fragment should be quite different.

```
for ($i = 0; $i <= ELEMENTS_OF_B - 1; $i++) {
    $new[ELEMENTS_OF_A + $i] = $a[$i];
}
```

The final PHP script is as follows.

file_37.1-3.php

```
<?php
define("ELEMENTS_OF_A", 20);
define("ELEMENTS_OF_B", 30);
define("ELEMENTS_OF_NEW", ELEMENTS_OF_A + ELEMENTS_OF_B);

//Read arrays $a and $b
$a = array();
for ($i = 0; $i <= ELEMENTS_OF_A - 1; $i++) {
    $a[$i] = trim(fgets(STDIN));
}
$b = array();
for ($i = 0; $i <= ELEMENTS_OF_B - 1; $i++) {
    $b[$i] = trim(fgets(STDIN));
}

//Create array $new
$new = array();
for ($i = 0; $i <= ELEMENTS_OF_A - 1; $i++) {
    $new[$i] = $a[$i];
}
```

```
for ($i = 0; $i <= ELEMENTS_OF_B - 1; $i++) {
    $new[ELEMENTS_OF_A + $i] = $b[$i];
}

//Display array $new
for ($i = 0; $i <= ELEMENTS_OF_NEW - 1; $i++) {
    echo $new[$i], "\t";
}

?>
```

Exercise 37.1-4 Merging Two-Dimensional Arrays

Write a PHP script that lets the user enter numerical values into arrays *\$a* and *\$b* of 10×20 and 30×20 elements respectively. Then, the script should create a new array *\$new* of 40×20 elements. In this new array, the first 10 rows should contain the elements of array *\$a* and the next 30 rows should contain the elements of array *\$b*.

Solution

This exercise can be solved pretty much the same way as the previous one. Below you can find the proposed solution. The approach of iterating through rows has been chosen, but iterating through columns could have been used as well.

file_37.1-4.php

```
<?php
define("COLUMNS", 20);
define("ROWS_OF_A", 10);
define("ROWS_OF_B", 30);
define("ROWS_OF_NEW", ROWS_OF_A + ROWS_OF_B);

//Read array $a
$a = array(array());
for ($i = 0; $i <= ROWS_OF_A - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

//Read array $b
$b = array(array());
for ($i = 0; $i <= ROWS_OF_B - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $b[$i][$j] = trim(fgets(STDIN));
    }
}
```

```

    }
}

//Create array $new
$new = array(array());
for ($i = 0; $i <= ROWS_OF_A - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $new[$i][$j] = $a[$i][$j];
    }
}
for ($i = 0; $i <= ROWS_OF_B - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $new[ROWS_OF_A + $i][$j] = $b[$i][$j];
    }
}

//Display array $new
for ($i = 0; $i <= ROWS_OF_NEW - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        echo $new[$i][$j], "\t";
    }
    echo "\n";
}
?>

```

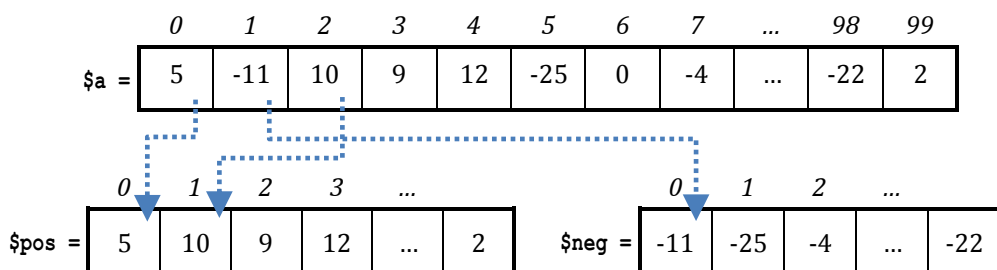
Exercise 37.1-5 *Creating Two Arrays – Separating Positive from Negative Values*

Write a PHP script that lets the user enter 100 numerical values into an array and then creates two new arrays, `$pos` and `$neg`. Array `$pos` should contain positive values, whereas array `$neg` should contain the negative ones. The value 0 (if any) should not be added to either of the final arrays, `$pos` or `$neg`.

Next, design the corresponding flowchart fragment only for that part of your script that creates the arrays `$pos` and `$neg`.

Solution

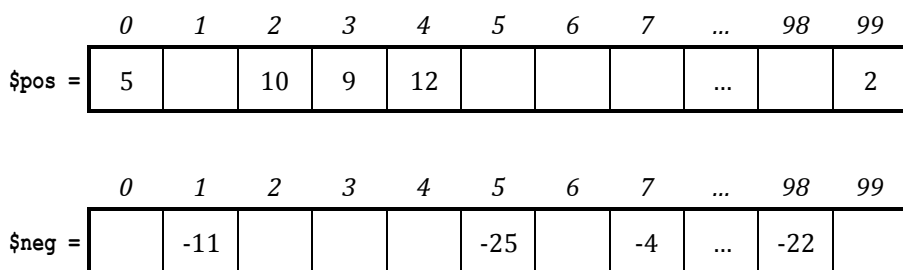
Let's analyze this exercise using the following example.



In this exercise, there is no one-to-one match between the index positions of the elements of array `$a` and the arrays `$pos` and `$neg`. For example, the element from position 1 of array `$a` is not stored in position 1 of array `$neg`, the element from position 2 of array `$a` is not stored in position 2 of array `$pos`, and so on. Thus, you **cannot** do the following,

```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($a[$i] > 0) {
        $pos[$i] = $a[$i];
    }
    elseif ($a[$i] < 0) {
        $neg[$i] = $a[$i];
    }
}
```

because it will result in the following two arrays.



What you need here are two independent index variables: `$pos_index` for the array `$pos`, and `$neg_index` for the array `$neg`. These index variables must be incremented independently, and only when an element is added to the corresponding array. The index variable `$pos_index` must be incremented only when an element is added to the array `$pos`, and the index variable `$neg_index` must be incremented only when an element is added to the array `$neg`.

```
$pos_index = 0;
$neg_index = 0;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($a[$i] > 0) {
        $pos[$pos_index] = $a[$i];
        $pos_index++;
    }
    elseif ($a[$i] < 0) {
        $neg[$neg_index] = $a[$i];
        $neg_index++;
    }
}
```

```

        $pos_index++;
    }
    elseif ($a[$i] < 0) {
        $neg[$neg_index] = $a[$i];
        $neg_index++;
    }
}

```

When this loop finishes iterating, the two arrays become as follows.

	0	1	2	3	...		0	1	2	...		
\$pos =	5	10	9	12	...	2	\$neg =	-11	-25	-4	...	-22

Notice: Please note that variables `$pos_index` and `$neg_index` have dual roles. When the loop iterates, each points to the next position in which a new element should be placed. When the loop finishes iterating, variables `$pos_index` and `$neg_index` actually contain the total number of elements in each corresponding array!

The complete solution is presented next.

file_37.1-5.php

```

<?php
define("ELEMENTS", 100);

$a = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $a[$i] = trim(fgets(STDIN));
}

//Create arrays $pos and $neg
$pos = array();
$neg = array();
$pos_index = 0;
$neg_index = 0;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($a[$i] > 0) {
        $pos[$pos_index] = $a[$i];
        $pos_index++;
    }
    elseif ($a[$i] < 0) {
        $neg[$neg_index] = $a[$i];
        $neg_index++;
    }
}

```

```

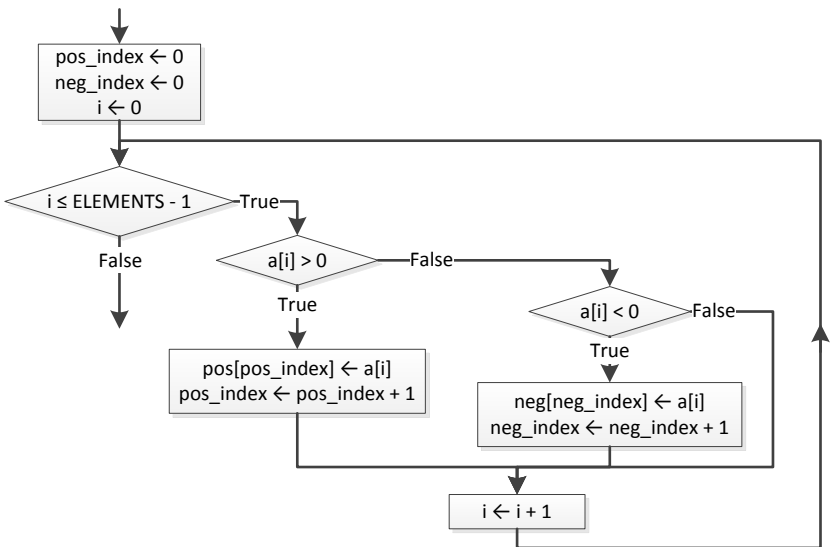
    }
}

for ($i = 0; $i <= $pos_index - 1; $i++) {
    echo $pos[$i], "\t";
}
echo "\n";
for ($i = 0; $i <= $neg_index - 1; $i++) {
    echo $neg[$i], "\t";
}
?>

```

Notice: Please note that the arrays `$pos` and `$neg` contain a total number of `$pos_index` and `$neg_index` elements respectively. This is why the two last loop control structures iterate until variable `$i` reaches values `$pos_index - 1` and `$neg_index - 1`, respectively, and not until `ELEMENTS - 1`, as you may mistakenly expect. Obviously the sum of `$pos_index + $neg_index` equals to `ELEMENTS`.

Now, let's design the corresponding flowchart fragment only for that part of the script that creates the arrays `$pos` and `$neg`. As you may notice, there is a multiple-alternative decision structure nested within a counted loop structure. The flowchart fragment is as follows.

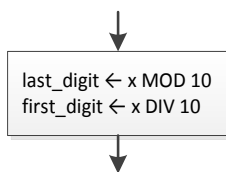


Exercise 37.1-6 Creating an Array with Those who Contain Digit 5

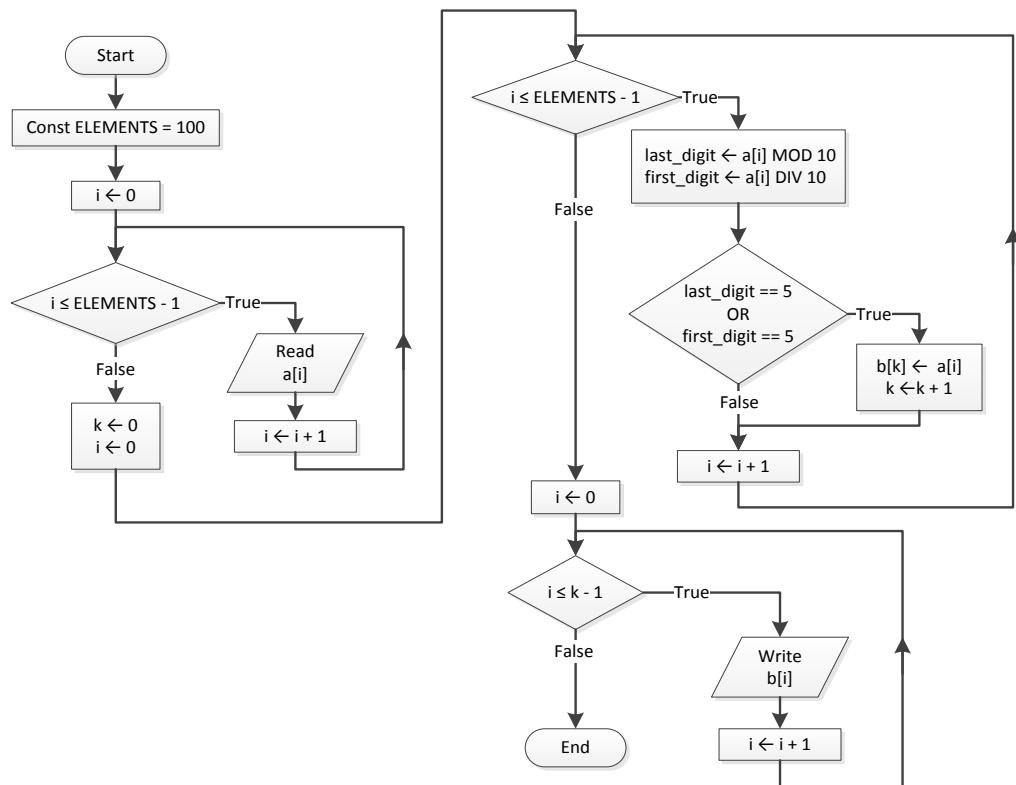
Write a PHP script that lets the user enter 100 two-digit integers into an array and then creates a new array of only the integers that contain at least one of the digit 5.

Solution

This exercise requires some knowledge from the past. In Chapter 13 you learned how to use the quotient and the remainder to split an integer into its individual digits. Here, the integer has two digits; therefore, you can use the following flowchart fragment to split any two-digit integer contained in variable x .



The final flowchart is as follows.



The corresponding PHP script is as follows.

file_37.1-6.php

```
<?php
define("ELEMENTS", 100);

$a = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $a[$i] = trim(fgets(STDIN));
}

$b = array();
$k = 0;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $last_digit = $a[$i] % 10;
    $first_digit = intval($a[$i] / 10);

    if ($last_digit == 5 || $first_digit == 5) {
        $b[$k] = $a[$i];
        $k++;
    }
}

for ($i = 0; $i <= $k - 1; $i++) {
    echo $b[$i], "\t";
}

?>
```

Remember! When you need to create a new array using values from an old array and there is no one-to-one match between their index positions, you need an extra variable that becomes the index for the new array. Of course, this variable should increase by 1 only when a new element is added into the new array. Moreover, when the loop that creates the new array finishes iterating, the value of this variable matches the total number of elements in the new array!

37.2 Data Validation with Arrays

As you already know, data validation is the process of restricting data input, which forces the user to enter only valid values. As you have already been taught in Chapter 31, there are three approaches that you can use depending on whether or not you wish to display an error message and whether you wish to display individual error messages for each error or just a single error message for any kind of error. Let's see how those three approaches can be adapted and used with arrays.

First Approach – Validating data input without error messages

In Chapter 31 you learned how to validate one single value entered by the user without displaying any error messages. For your convenience, the code fragment given in general form is presented once again.

```
do {  
    echo "Prompt message";  
    $input_data = trim(fgets(STDIN));  
} while ($input_data test 1 fails || $input_data test 2 fails || ...);
```

Do you remember how this operates? If the user persistently enters an invalid value, the main idea is to prompt him or her repeatedly, until he or she gets bored and finally enters a valid one. Of course, if the user enters a valid value right from the beginning, the flow of execution simply continues to the next part of the script.

You can use the same principle when entering data into arrays. If you use a counted loop structure to iterate for all elements of the array, the code fragment becomes as follows.

```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    do {  
        echo "Prompt message";  
        $input_data = trim(fgets(STDIN));  
    } while ($input_data test 1 fails || $input_data test 2 fails || ...);  
    $input_array[$i] = $input_data;  
}
```

As you can see, when the flow of execution exits the post-test loop structure, the variable `$input_data` definitely contains a valid value which in turn is assigned to a position of the array `$input_array`. However, the same process can be implemented more simply, without using the extra variable `$input_data`, as follows.

```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    do {  
        echo "Prompt message";  
        $input_array[$i] = trim(fgets(STDIN));  
    } while ($input_array[$i] test 1 fails ||  
            $input_array[$i] test 2 fails || ...);  
}
```

Second Approach – Validating data input with one single error message

As previously, the next code fragment is taken from Chapter 31 and adapted to operate with an array. It validates data input and displays an error message (that is, the same error message for any type of input error).

```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    echo "Prompt message";  
    $input_array[$i] = trim(fgets(STDIN));
```

```
while($input_array[$i] test 1 fails || $input_array[$i] test 2 fails || ...) {  
    echo "Error message \n";  
    echo "Prompt message";  
    $input_array[$i] = trim(fgets(STDIN));  
}  
}
```

Third Approach – Validating data input with individual error messages

Once again, the next code fragment is taken from Chapter 31 and adapted to operate with an array. It validates data input and displays an error message (that is, individual error messages, one for each type of input error).

```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    do {  
        echo "Prompt message";  
        $input_array[$i] = trim(fgets(STDIN));  
        $failure = false;  
        if ($input_array[$i] test 1 fails) {  
            echo "Error message 1 \n";  
            $failure = true;  
        }  
        elseif ($input_array[$i] test 2 fails) {  
            echo "Error message 2 \n";  
            $failure = true;  
        }  
        elseif (...  
            ...  
        )  
    } while ($failure == true);  
}
```

Exercise 37.2-1 *Displaying Odds in Reverse Order – Validation Without Error Messages*

Write a PHP script that prompts the user to enter 20 odd integers into an array and then displays them in reverse order. The script should also validate data input, allowing the user to enter only odd integers. There is no need to display any error messages.

Solution

As the wording of the exercise implies, not all entered values should be added in the array—only the odd ones. The PHP script is as follows.

file_37.2-1.php

```
<?php
define("ELEMENTS", 20);

$odds = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    do {
        echo "Enter an odd integer: ";
        $odds[$i] = trim(fgets(STDIN));
    } while ($odds[$i] % 2 == 0);
}

//display elements backwards
for ($i = ELEMENTS - 1; $i >= 0; $i--) {
    echo $odds[$i], "\t";
}

?>
```

Data input and validation

Exercise 37.2-2 *Displaying Odds in Reverse Order – Validation with One Error Message*

Write a PHP script that prompts the user to enter 20 odd integers into an array and then displays them in reverse order. The script should also validate data input and display an error message when the user enters any non-numeric values or any even integers.

Solution

This is the same as you've previously seen, except for data validation. The second approach is used, according to which an error message is displayed when the user enters even integers. The PHP script is presented next.

file_37.2-2.php

```
<?php
define("ELEMENTS", 20);

$odds = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    echo "Enter an odd integer: ";
    $odds[$i] = trim(fgets(STDIN));
    while (is_numeric($odds[$i]) != true || $odds[$i] % 2 == 0) {
        echo "Invalid value! \n";
        echo "Enter an odd integer: ";
    }
}
```

Data input and validation


```

    $odds[$i] = trim(fgets(STDIN));
}
}

//display elements backwards
for ($i = ELEMENTS - 1; $i >= 0; $i--) {
    echo $odds[$i], "\t";
}
?>

```

Exercise 37.2-3 *Displaying Odds in Reverse Order – Validation with Individual Error Messages*

Write a PHP script that prompts the user to enter 20 odd integers in an array and then displays them in reverse order. The script should also validate data input and display individual error messages when the user enters any non-numeric values, any even integers, or any reals.

Solution

The third approach is used, according to which individual error messages are displayed when the user enters non-numeric values, even integers, or reals. The PHP script is as follows.

file_37.2-3.php

```

<?php
define("ELEMENTS", 20);

$odds = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    do {
        echo "Enter an odd integer: ";
        $odds[$i] = trim(fgets(STDIN));
        $failure = false;
        if (is_numeric($odds[$i]) != true) {
            echo "Please enter a numeric value! \n";
            $failure = true;
        }
        elseif ($odds[$i] != intval($odds[$i])) {
            echo "Please enter an integer! \n";
            $failure = true;
        }
        elseif ($odds[$i] % 2 == 0) {

```

Data input and validation

```
        echo "Please enter an odd! \n";
        $failure = true;
    }
} while ($failure == true);
}

//display elements backwards
for ($i = ELEMENTS - 1; $i >= 0; $i--) {
    echo $odds[$i], "\t";
}
?>
```

37.3 Finding Minimum and Maximum Values in Arrays

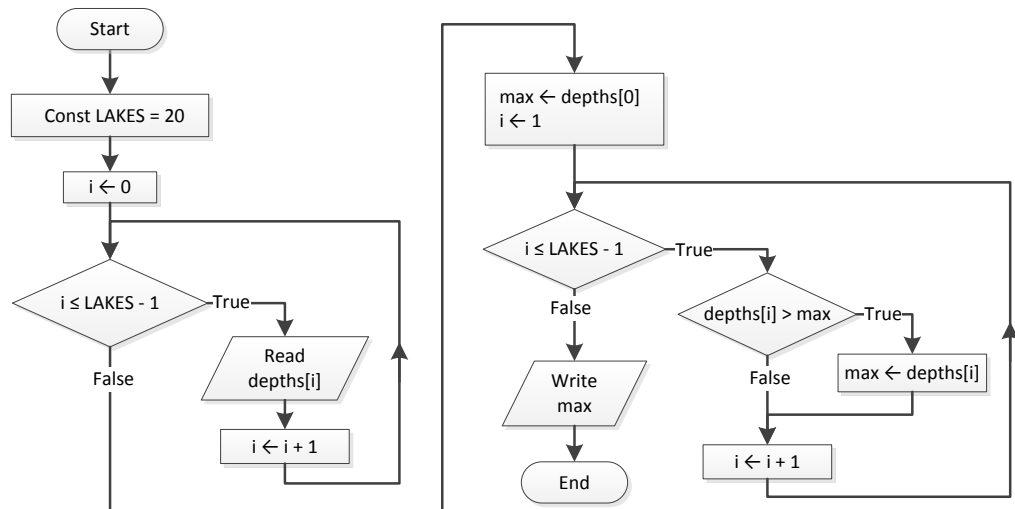
This is the third and last time that this subject is raked up in this book. The first time was in Section 4 (Decision Control Structures) and the second time was in Section 5 (Loop Control Structures). So, there is not much left to discuss except that when you want to find the minimum or maximum value of an array that already contains some values, you needn't worry about the initial values of variables `$min` or `$max` because you can just assign to them the value of the first element of the array!

Exercise 37.3-1 Which Depth is the Greatest?

Design a flowchart and write the corresponding PHP script that lets the user enter the depths of 20 lakes and then displays the depth of the deepest one.

Solution

After the user enters the depths of the 20 lakes in the array `$depths`, the initial value of variable `$max` can be set as the value of the first element of array `$depths`, that is `$depths[0]`. The script can then search for any value greater than this. The final solution is quite simple and is presented next without further explanation.



The corresponding PHP script is as follows.

file_37.3-1.php

```
<?php
define("LAKES", 20);

$depths = array();
for ($i = 0; $i <= LAKES - 1; $i++) {
    $depths[$i] = trim(fgets(STDIN));
}

//initial value
$max = $depths[0];
//Search furthermore, starting from position 1
for ($i = 1; $i <= LAKES - 1; $i++) {
    if ($depths[$i] > $max) {
        $max = $depths[$i];
    }
}
echo $max;

?>
```

Notice: It wouldn't be wrong to start iterating from position 0 instead of 1, though the script would perform one useless iteration.

Notice: *It wouldn't be wrong to assign an "almost arbitrary" initial value to variable $\$max$ but there is no reason to do so. The value of the first element is just fine! If you insist though, you can assign an initial value of 0, since there is no lake on planet Earth with a negative depth.*

Exercise 37.3-2 Which Lake is the Deepest?

Write a PHP script that lets the user enter the names and the depths of 20 lakes and then displays the name of the deepest one.

Solution

In this exercise, you need two one-dimensional arrays: one to hold the names, and one to hold the depths of the lakes. The solution is presented next.

file_37.3-2.php

```
<?php
define("LAKES", 20);

$names = array();
$depths = array();
for ($i = 0; $i <= LAKES - 1; $i++) {
    $names[$i] = trim(fgets(STDIN));
    $depths[$i] = trim(fgets(STDIN));
}

$max = $depths[0];
$m_name = $names[0];
for ($i = 1; $i <= LAKES - 1; $i++) {
    if ($depths[$i] > $max) {
        $max = $depths[$i];
        $m_name = $names[$i];
    }
}

echo $m_name;
?>
```

Exercise 37.3-3 Which Lake, in Which Country, Having Which Average Area, is the Deepest?

Write a PHP script that lets the user enter the names and the depths of 20 lakes as well as the country in which they belong, and their average area. The script should then display all available information about the deepest lake.

Solution

In this exercise, you need four one-dimensional arrays: one to hold the names, one to hold the depths, one to hold the names of the countries, and one to hold the lakes' average areas. There are two approaches, actually. The first one is pretty much the same as the one used in the previous exercise. The second one is more efficient since it uses fewer variables. Let's study them both!

First Approach – One variable for each

This is pretty much the same as the one used in the previous exercise. The solution is presented next.

file_37.3-3a.php

```
<?php
define("LAKES", 20);

$names = array();
$depths = array();
$countries = array();
$areas = array();
for ($i = 0; $i <= LAKES - 1; $i++) {
    $names[$i] = trim(fgets(STDIN));
    $depths[$i] = trim(fgets(STDIN));
    $countries[$i] = trim(fgets(STDIN));
    $areas[$i] = trim(fgets(STDIN));
}

$max = $depths[0];
$m_name = $names[0];
$m_country = $countries[0];
$m_area = $areas[0];
for ($i = 1; $i <= LAKES - 1; $i++) {
    if ($depths[$i] > $max) {
        $max = $depths[$i];
        $m_name = $names[$i];
        $m_country = $countries[$i];
        $m_area = $areas[$i];
    }
}

echo $max, $m_name, $m_country, $m_area;
?>
```

Second Approach – One index for everything

In this approach fewer variables are used. Instead of using all those variables (\$m_name, \$m_country, and \$m_area), you can use just one variable, a variable that holds the index in which the maximum value exists!

Confused? Let’s look at the next example of six lakes. The depths are expressed in feet and the average areas in square miles.

\$names =	0	Toba	\$depths =	1660	\$countries =	Indonesia	\$areas =	440
	1	Issyk Kul		2192		Kyrgyzstan		2408
	2	Baikal		5380		Russia		12248
	3	Crater		1950		USA		21
	4	Karakul		750		Tajikistan		150
	5	Quesnel		2000		Canada		103

Obviously the deepest lake is Lake Baikal at position 2. However, instead of holding the name “Baikal” in variable \$m_name, the country “Russia” in variable \$m_country, and the area “12248” in variable \$m_area as in the previous approach, you can use just one variable to hold the index position in which these values actually exist (in this case, this is value 2). The solution is presented next.

file_37.3-3b.php

```
<?php
define("LAKES", 20);

$names = array();
$depths = array();
$countries = array();
$areas = array();
for ($i = 0; $i <= LAKES - 1; $i++) {
    $names[$i] = trim(fgets(STDIN));
    $depths[$i] = trim(fgets(STDIN));
    $countries[$i] = trim(fgets(STDIN));
    $areas[$i] = trim(fgets(STDIN));
}

$max = $depths[0];
$index_of_max = 0;
for ($i = 1; $i <= LAKES - 1; $i++) {
```

```

    if ($depths[$i] > $max) {
        $max = $depths[$i];
        $index_of_max = $i;
    }
}

echo $depths[$index_of_max], $names[$index_of_max];
echo $countries[$index_of_max], $areas[$index_of_max];
?>

```

Remember! Assigning an initial value of 0 to variable `$index_of_max` is necessary since there is always a possibility that the maximum value does exist in position 0.

Exercise 37.3-4 Which Students are the Tallest?

Write a PHP script that prompts the user to enter the names and the heights of 100 students and then displays the names of all those who share the one tallest height. Moreover, the script should validate data input and display an error message when the user enters any non-numeric or negative values.

Solution

In this exercise, a loop control structures should search for the greatest value, and afterwards, another loop control structure can search the array for all values that are equal to that greatest value.

The solution is presented next.

file_37.3-4.php

```

<?php
define("STUDENTS", 100);

$names = array();
$grades = array();
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "Enter name for student No: ", $i + 1;
    $names[$i] = trim(fgets(STDIN));

    echo "Enter his or her grade: ";
    $grades[$i] = trim(fgets(STDIN));
    while (is_numeric($grades[$i]) != true || $grades[$i] < 0) {
        echo "Invalid value! \n";
        echo "Enter his or her grade: ";
        $grades[$i] = trim(fgets(STDIN));
    }
}

```

```

    }
}

$max = $grades[0];
for ($i = 1; $i <= STUDENTS - 1; $i++) {
    if ($grades[$i] > $max) {
        $max = $grades[$i];
    }
}

echo "The following students have got the greatest grade:\n";
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    if ($grades[$i] == $max) {
        echo $names[$i], "\n";
    }
}

?>

```

Notice: Please note that this exercise could not have been solved without the use of an array.

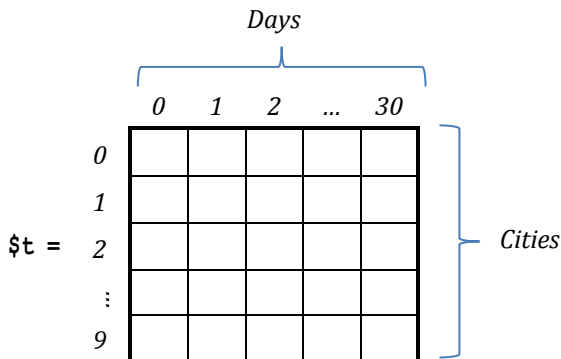
Exercise 37.3-5 Finding the Minimum Value of a Two-Dimensional Array

Write a PHP script that lets the user enter the temperatures (in degrees Fahrenheit) recorded at 12:00 p.m. on each day in January in 10 different cities. The PHP script should display the lowest temperature.

Next, design the corresponding flowchart fragment for only that part of the script that finds the lowest temperature.

Solution

In this exercise, you need the following array.



Notice: The array `$t` has 31 columns (0 to 30), as many as there are days in January.

There is nothing new here. The initial value of variable `$min` can be the value of the element `$t[0][0]`. Then, the script can iterate through rows, or even through columns, to search for the minimum value. The solution is presented next.

file_37.3-5.php

```
<?php
define("CITIES", 10);
define("DAYS", 31);

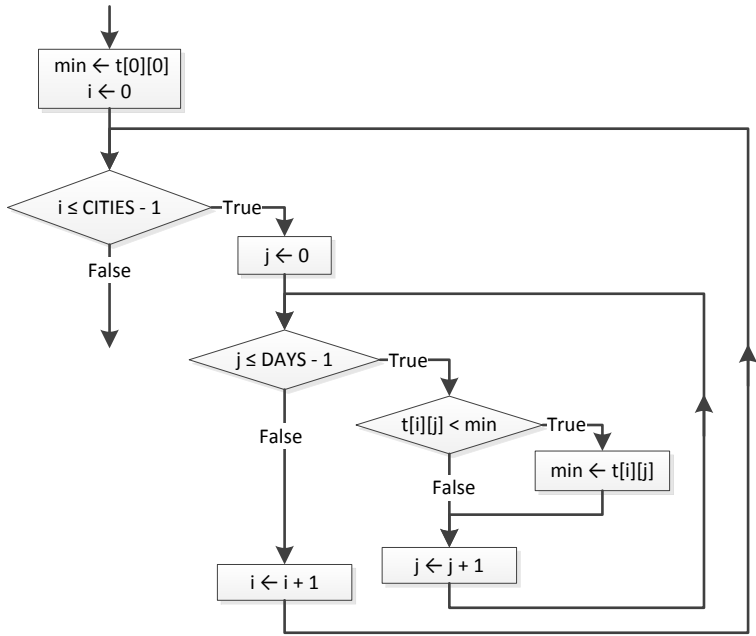
//Read array $t
$t = array(array());
for ($i = 0; $i <= CITIES - 1; $i++) {
    for ($j = 0; $j <= DAYS - 1; $j++) {
        $t[$i][$j] = trim(fgets(STDIN));
    }
}

//Find minimum
$min = $t[0][0];
for ($i = 0; $i <= CITIES - 1; $i++) {
    for ($j = 0; $j <= DAYS - 1; $j++) {
        if ($t[$i][$j] < $min) {
            $min = $t[$i][$j];
        }
    }
}

echo $min;

?>
```

Following is the corresponding flowchart fragment for the part of the script that finds the lowest temperature (minimum value of array `$t`).

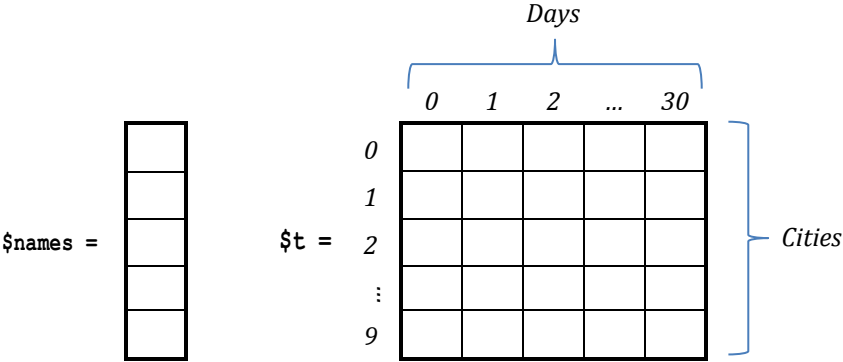


Exercise 37.3-6 *Finding the City with the Coldest Day*

Write a PHP script that lets the user enter the temperatures (in degrees Fahrenheit) recorded at 12:00 p.m. on each day in January in 10 different cities. The PHP script should display the name of the city that had the lowest temperature and on which day it was recorded.

Solution

In this exercise, the following two arrays are needed.



The solution is simple. Every time variable `$min` updates its value, two variables, `$m_i` and `$m_j`, can hold the values of variables `$i` and `$j` respectively. In the end, these two

variables will contain the row index and the column index of the position in which the maximum value exists. The solution is as follows.

file_37.3-6.php

```
<?php
define("CITIES", 10);
define("DAYS", 31);

$names = array();
$t = array(array());
for ($i = 0; $i <= CITIES - 1; $i++) {
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= DAYS - 1; $j++) {
        $t[$i][$j] = trim(fgets(STDIN));
    }
}

$min = $t[0][0];
$m_i = 0;
$m_j = 0;
for ($i = 0; $i <= CITIES - 1; $i++) {
    for ($j = 0; $j <= DAYS - 1; $j++) {
        if ($t[$i][$j] < $min) {
            $min = $t[$i][$j];
            $m_i = $i;
            $m_j = $j;
        }
    }
}

echo "Minimum temperature: ", $min, "\n";
echo "City: ", $names[$m_i], "\n";
echo "Day: ", $m_j + 1;

?>
```

Remember! Assigning an initial value of 0 to variables `$m_i` and `$m_j` is necessary since there is always a possibility that the maximum value is the value of the element `$t[0][0]`.

Exercise 37.3-7 *Finding the Minimum and the Maximum Value of Each Row*

Write a PHP script that lets the user enter values in a 20×30 array and then finds and displays the minimum and the maximum values of each row.

Solution

There are two approaches, actually. The first approach creates two auxiliary one-dimensional arrays, `$min` and `$max`, and then displays them. Arrays `$min` and `$max` will contain, in each position, the minimum and the maximum values of each row respectively. On the other hand, the second approach finds and directly displays the minimum and maximum values of each row. Let's study both approaches.

First Approach – Creating auxiliary arrays

To better understand this approach, let's use the “from inner to outer” method. When the following code fragment completes its iterations, the auxiliary one-dimensional arrays `$min` and `$max` contain at position 0 the minimum and the maximum values of the first row (row index 0) of array `$a` respectively. Assume variable `$i` contains value 0.

```
$min[$i] = $a[$i][0];
$max[$i] = $a[$i][0];
for ($j = 1; $j <= COLUMNS - 1; $j++) {
    if ($a[$i][$j] < $min[$i]) {
        $min[$i] = $a[$i][$j];
    }
    if ($a[$i][$j] > $max[$i]) {
        $max[$i] = $a[$i][$j];
    }
}
```

Notice: Please note that the initial value of variable `$j` is 1. It wouldn't be wrong to start iterating from column index 0 instead of 1, though the script would perform one useless iteration.

Now that everything has been clarified, in order to process the whole array `$a`, you can just nest this code fragment into a counted loop structure which iterates for all rows as shown next.

```
for ($i = 0; $i <= ROWS - 1; $i++) {
    $min[$i] = $a[$i][0];
    $max[$i] = $a[$i][0];
    for ($j = 1; $j <= COLUMNS - 1; $j++) {
        if ($a[$i][$j] < $min[$i]) {
            $min[$i] = $a[$i][$j];
        }
    }
}
```

```

    }
    if ($a[$i][$j] > $max[$i]) {
        $max[$i] = $a[$i][$j];
    }
}
}

```

The final PHP script is as follows.

file_37.3-7a.php

```

<?php
define("ROWS", 30);
define("COLUMNS", 20);

$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

$min = array();
$max = array();
for ($i = 0; $i <= ROWS - 1; $i++) {
    $min[$i] = $a[$i][0];
    $max[$i] = $a[$i][0];
    for ($j = 1; $j <= COLUMNS - 1; $j++) {
        if ($a[$i][$j] < $min[$i]) {
            $min[$i] = $a[$i][$j];
        }
        if ($a[$i][$j] > $max[$i]) {
            $max[$i] = $a[$i][$j];
        }
    }
}

for ($i = 0; $i <= ROWS - 1; $i++) {
    echo $min[$i], $max[$i];
}

?>

```

Second Approach – Finding and directly displaying minimum and maximum values

Let's use the “from inner to outer” method once again. The next code fragment finds and directly displays the minimum and the maximum values of the first row (row index 0) of array `$a`. Assume variable `$i` contains the value 0.

```
$min = $a[$i][0];
$max = $a[$i][0];
for ($j = 1; $j <= COLUMNS - 1; $j++) {
    if ($a[$i][$j] < $min) {
        $min = $a[$i][$j];
    }
    if ($a[$i][$j] > $max) {
        $max = $a[$i][$j];
    }
}

echo $min, $max;
```

Now that everything has been clarified, in order to process the whole array `$a`, you can just nest this code fragment into a counted loop structure which iterates for all rows, as follows.

```
for ($i = 0; $i <= ROWS - 1; $i++) {
    $min = $a[$i][0];
    $max = $a[$i][0];
    for ($j = 1; $j <= COLUMNS - 1; $j++) {
        if ($a[$i][$j] < $min) {
            $min = $a[$i][$j];
        }
        if ($a[$i][$j] > $max) {
            $max = $a[$i][$j];
        }
    }

    echo $min, $max;
}
```

The final PHP script is as follows.

file_37.3-7b.php

```
<?php
define("ROWS", 30);
define("COLUMNS", 20);
```

```

$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

for ($i = 0; $i <= ROWS - 1; $i++) {
    $min = $a[$i][0];
    $max = $a[$i][0];
    for ($j = 1; $j <= COLUMNS - 1; $j++) {
        if ($a[$i][$j] < $min) {
            $min = $a[$i][$j];
        }
        if ($a[$i][$j] > $max) {
            $max = $a[$i][$j];
        }
    }

    echo $min, $max;
}
?>

```

Exercise 37.3-8 *Finding the Minimum and the Maximum Value of Each Column*

Write a PHP script that lets the user enter values into a 20×30 array and then finds and displays the minimum and the maximum values of each column.

Solution

As in the previous exercise, there are two approaches.

First Approach – Creating auxiliary arrays

When the following code fragment completes its iterations, the auxiliary one-dimensional arrays `$min` and `$max` contain at position 0 the minimum and the maximum values of the first column (column index 0) of array `$a` respectively. Assume variable `$j` contains the value 0.

```

$min[$j] = $a[0][$j];
$max[$j] = $a[0][$j];
for ($i = 1; $i <= ROWS - 1; $i++) {
    if ($a[$i][$j] < $min[$j]) {

```

```

        $min[$j] = $a[$i][$j];
    }
    if ($a[$i][$j] > $max[$j]) {
        $max[$j] = $a[$i][$j];
    }
}
}

```

Notice: Please note that the initial value of variable `$i` is 1. It wouldn't be wrong to start iterating from row index 0 instead of 1, though the script would perform one useless iteration.

To process the whole array `$a`, you can just nest this code fragment into a counted loop structure that iterates for all columns as follows.

```

for ($j = 0; $j <= COLUMNS - 1; $j++) {
    $min[$j] = $a[0][$j];
    $max[$j] = $a[0][$j];
    for ($i = 1; $i <= ROWS - 1; $i++) {
        if ($a[$i][$j] < $min[$j]) {
            $min[$j] = $a[$i][$j];
        }
        if ($a[$i][$j] > $max[$j]) {
            $max[$j] = $a[$i][$j];
        }
    }
}
}

```

Notice: Finding the minimum and maximum values of each column is as easy as finding the minimum and the maximum value of each row, as you learned in the previous exercise. Please note that the two counted loop structures of `$i` and `$j` have switched places.

The final PHP script is as follows.

file_37.3-8a.php

```

<?php
define("ROWS", 30);
define("COLUMNS", 20);

$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

```



```

}

$min = array();
$max = array();
for ($j = 0; $j <= COLUMNS - 1; $j++) {
    $min[$j] = $a[0][$j];
    $max[$j] = $a[0][$j];
    for ($i = 1; $i <= ROWS - 1; $i++) {
        if ($a[$i][$j] < $min[$j]) {
            $min[$j] = $a[$i][$j];
        }
        if ($a[$i][$j] > $max[$j]) {
            $max[$j] = $a[$i][$j];
        }
    }
}

for ($j = 0; $j <= COLUMNS - 1; $j++) {
    echo $min[$j], $max[$j];
}

?>

```

Second Approach – Finding and directly displaying minimum and maximum values

Without further clarification, and using the second approach of the previous exercise as a guide, the corresponding solution is as follows.

file_37.3-8b.php

```

<?php
define("ROWS", 30);
define("COLUMNS", 20);

$a = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $a[$i][$j] = trim(fgets(STDIN));
    }
}

for ($j = 0; $j <= COLUMNS - 1; $j++) {
    $min = $a[0][$j];

```

```
$max = $a[0][$j];  
for ($i = 1; $i <= ROWS - 1; $i++) {  
    if ($a[$i][$j] < $min) {  
        $min = $a[$i][$j];  
    }  
    if ($a[$i][$j] > $max) {  
        $max = $a[$i][$j];  
    }  
}  
echo $min, $max;  
}  
?>
```

37.4 Sorting Arrays

Sorting algorithms are an important topic in computer science. A sorting algorithm is an algorithm that puts elements of an array in a certain order. There are many sorting algorithms and each one of them has particular strengths and weaknesses.

Most sorting algorithms work by comparing the elements of the array. They are usually evaluated by their efficiency and their memory requirements.

There are many sorting algorithms. Some of them are:

- the bubble sort algorithm
- the modified bubble sort algorithm
- the selection sort algorithm
- the insertion sort algorithm
- the heap sort algorithm
- the merge sort algorithm
- the quicksort algorithm

As regards their efficiency, the bubble sort algorithm is considered the least efficient, while each succeeding algorithm in the list performs better than the preceding one. The quicksort algorithm is considered one of the best and fastest sorting algorithms, especially for large scale data operations.

Sorting algorithms can be used for more than just displaying data in ascending or descending order. For example, if an exercise requires you to display the three biggest numbers of an array, the PHP script can sort the array in descending order and then display only the first three elements, that is, those at index positions 0, 1, and 2.

Sorting algorithms can also help you find the minimum and the maximum values from a set of given values. If an array is sorted in ascending order, the minimum value exists at the first index position and the maximum value exists at the last index position. Of course, it is very inefficient to sort an array so as to find minimum and maximum values; you have already learned a more efficient method. But if for some

reason an exercise requires that an array be sorted and you need the minimum or the maximum value, you know where you can find them!

Exercise 37.4-1 *The Bubble Sort Algorithm – Sorting One-Dimensional Arrays with Numeric Values*

Write a PHP script that lets the user enter 20 numerical values into an array and then sorts them in ascending order using the bubble sort algorithm.

Solution

The bubble sort algorithm is probably one of the most inefficient sorting algorithms but it is widely used for teaching purposes. The main idea (when asked to sort an array in ascending order) is to repeatedly move the smallest elements of the array to the positions of lowest index. This works as follows: the algorithm iterates through the elements of the array, compares each pair of adjacent elements, and then swaps their contents (if they are in the wrong order). This process is repeated many times until the array is sorted.

For example, let’s try to sort the following array in ascending order.

A =

17	0
25	1
8	2
5	3
49	4
12	5

The lowest value is the value 5. According to the bubble sort algorithm, this value should gradually “bubble” or “rise” to position 0, like bubbles rising in a glass of cola. When the value 5 has been moved into position 0, the next smallest value is the value 8. Now, the value 8 should “bubble” to position 1. Next is the value 12, which should “bubble” to position 2, and so on. This process repeats until all elements are placed in proper position.

But how can this “bubbling” be done using an algorithm? Let’s see the whole process in more detail. For the previous array A of six elements, five passes must be performed.

First Pass

1st Compare

Initially, elements at index positions 4 and 5 are compared. Since the value 12 is less than the value 49, these two elements swap their content.

2nd Compare

Elements at index positions 3 and 4 are compared. Since the value 12 is **not** less than the value 5, **no** swapping is done.

3rd Compare

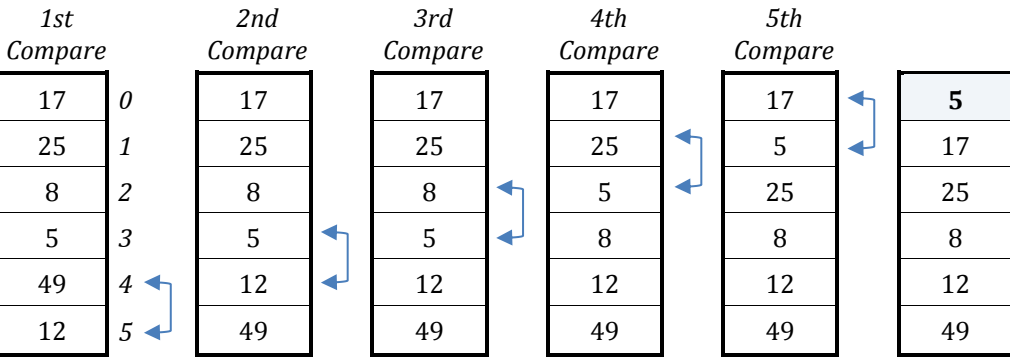
Elements at index positions 2 and 3 are compared. Since the value 5 is less than the value 8, these two elements swap their content.

4th Compare

Elements at index positions 1 and 2 are compared. Since the value 5 is less than the value 25, these two elements swap their content.

5th Compare

Elements at index positions 0 and 1 are compared. Since the value 5 is less than the value 17, these two elements swap their content.



The first pass has been completed but, as you can see, the array has not been sorted yet. The only value that has actually been placed in proper position is the value 5. However, since more passes will follow, there is no need for the value 5 to take part in the subsequent compares. In the pass that follows, one less compare will be performed—that is, four compares.

Second Pass

1st Compare

Elements at index positions 4 and 5 are compared. Since the value 49 is **not** less than the value 12, **no** swapping is done.

2nd Compare

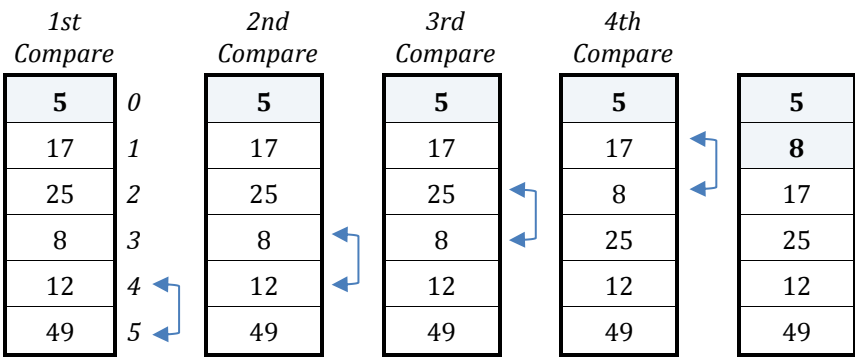
Elements at index positions 3 and 4 are compared. Since the value 12 is **not** less than the value 8, **no** swapping is done.

3rd Compare

Elements at index positions 2 and 3 are compared. Since the value 8 is less than the value 25, these two elements swap their content.

4th Compare

Elements at index positions 1 and 2 are compared. Since the value 8 is less than the value 17, these two elements swap their content.



The second pass has been completed and the value of 8 has been placed in proper position. However, since more passes will follow, there is no need for the value 8 (nor 5, of course) to take part in the subsequent compares. In the pass that follows, one less compare will be performed—that is, three compares.

Third Pass

1st Compare

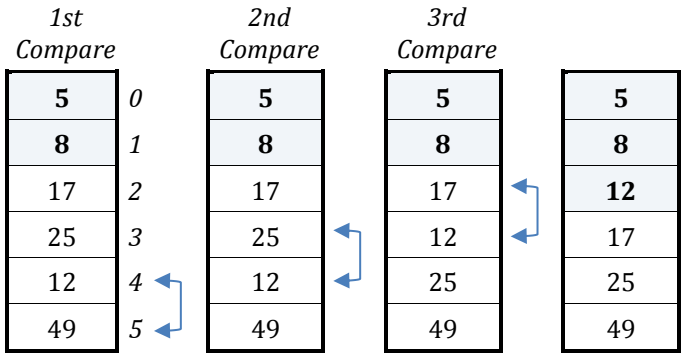
Elements at index positions 4 and 5 are compared. Since the value 49 is **not** less than the value 12, **no** swapping is done.

2nd Compare

Elements at index positions 3 and 4 are compared. Since the value 12 is less than the value 25, these two elements swap their content.

3rd Compare

Elements at index positions 2 and 3 are compared. Since the value 12 is less than the value 17, these two elements swap their content.



The third pass has been completed and the value of 12 has been placed in proper position. As previously, since more passes will follow there is no need for the value 12 (nor the values 5 and 8, of course) to take part in the subsequent compares. In the pass that follows, one compare less will be performed—that is, two compares.

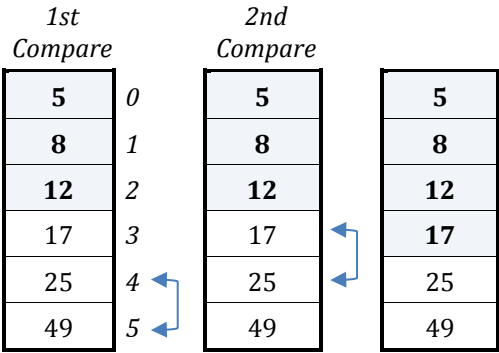
Fourth Pass

1st Compare

Elements at index positions 4 and 5 are compared. Since the value 49 is **not** less than the value 25, **no** swapping is done.

2nd Compare

Elements at index positions 3 and 4 are compared. Since the value 25 is **not** less than the value 17, **no** swapping is done.

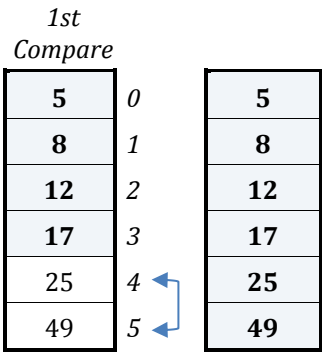


The fourth pass has been completed and the value 17 has been placed in proper position. As previously, since one last pass will follow, there is no need for the value 17 (nor the values 5, 8, and 12, of course) to take part in the subsequent compares. In the last pass that follows, one compare less will be performed—that is one compare.

Fifth pass

1st Compare

Elements at index positions 4 and 5 are compared. Since the value 49 is **not** less than the value 12, **no** swapping is done.



The fifth pass has been completed and the final two values (25 and 49) have been placed in proper position. The bubble sort algorithm has finished and the array is sorted in ascending order!

Now you need a PHP script that can do the whole previous process. Let’s use the “from inner to outer” method. The code fragment that performs the first pass is

shown below. Please note that this is the inner (nested) loop control structure. Assume variable `$m` contains the value 1.

```
for ($n = ELEMENTS - 1; $n >= $m; $n--) {
    if ($a[$n] < $a[$n - 1]) {
        $temp = $a[$n];
        $a[$n] = $a[$n - 1];
        $a[$n - 1] = $temp;
    }
}
```

Notice: In the first pass, variable `$m` must contain the value 1. This assures that at the last iteration, the elements that are compared are those at positions 1 and 0.

Remember! Swapping the contents of two elements uses a method you have already learned! Please recall the two glasses of orange juice and lemon juice. If this doesn't ring a bell, you need to refresh your memory and reread the corresponding exercise in Chapter 8.

The second pass can be performed if you just re-execute the previous code fragment. Variable `$m`, however, needs to contain the value 2. This will ensure that the element at position 0 won't be compared again. Similarly, for the third pass, the previous code fragment can be re-executed but variable `$m` needs to contain the value 3 for the same reason.

Accordingly, the previous code fragment needs to be executed five times (one for each pass), and each time variable `$m` must be incremented by 1. The final code fragment that sorts array `$a` using the bubble sort algorithm is as follows.

```
for ($m = 1; $m <= ELEMENTS - 1; $m++) {
    for ($n = ELEMENTS - 1; $n >= $m; $n--) {
        if ($a[$n] < $a[$n - 1]) {
            $temp = $a[$n];
            $a[$n] = $a[$n - 1];
            $a[$n - 1] = $temp;
        }
    }
}
```

The complete PHP script is as follows.

file_37.4-1.php

```
<?php
define("ELEMENTS", 20);

$a = array();

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
```

```

    $a[$i] = trim(fgets(STDIN));
}

for ($m = 1; $m <= ELEMENTS - 1; $m++) {
    for ($n = ELEMENTS - 1; $n >= $m; $n--) {
        if ($a[$n] < $a[$n - 1]) {
            $temp = $a[$n];
            $a[$n] = $a[$n - 1];
            $a[$n - 1] = $temp;
        }
    }
}

for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    echo $a[$i], "\t";
}

?>

```

Notice: The bubble sort algorithm is very inefficient. The total number of compares that it performs is $\frac{N(N-1)}{2}$, where N is the total number of array elements.

Notice: The total number of swaps depends on the given array. The worst case is when you want to sort in descending order an array that is already sorted in ascending order, or vice versa.

Exercise 37.4-2 *Sorting One-Dimensional Arrays with Alphanumeric Values*

Write a code fragment that sorts the alphanumeric values of an array in descending order using the bubble sort algorithm.

Solution

Comparing this exercise to the previous one, two things are different. First, the bubble sort algorithm needs to sort alphanumeric values, such as names of people or names of cities; and second, it has to sort them in descending order.

Don't worry though, because there is some good news for you! In order to sort alphanumeric data, you don't have to change anything in the algorithm! PHP handles letters the same way it handles numbers. The letter "A" is considered "less than" the letter "B", the letter "B" is considered "less than" the letter "C," and so on. Of course, if the array contains words in which the first letter is identical, PHP moves on and compares their second letters and perhaps their third letter (if there is need to do so). For example, the name "John" is considered "less than" the name "Jonathan" because the third letter "h" is "less than" the third letter "n."

Notice: When you are thinking of alphanumeric sorting, think of your dictionary and how words are sorted there.

And now let's see what you need to change so that the algorithm can sort in descending order. Do you remember how the bubble sort algorithm actually works? Smaller elements gradually "bubble" to positions of lowest index, like bubbles rise in a glass of cola. What you want in this exercise is to make the bigger (instead of the smaller) elements "bubble" to lower index positions. Therefore, all you need to do is simply reverse the comparison operator of the decision control structure!

The code fragment that sorts alphanumeric, and of course numeric, values in descending order is as follows.

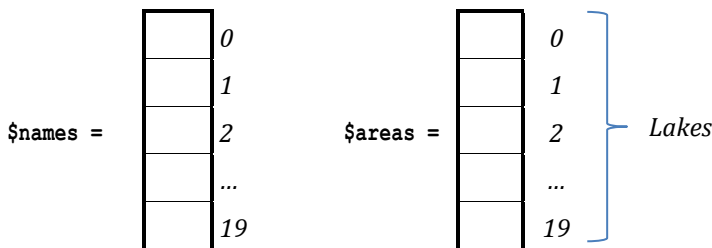
```
for ($m = 1; $m <= ELEMENTS - 1; $m++) {
    for ($n = ELEMENTS - 1; $n >= $m; $n--) {
        if ($a[$n] > $a[$n - 1]) {
            $temp = $a[$n];
            $a[$n] = $a[$n - 1];
            $a[$n - 1] = $temp;
        }
    }
}
```

Exercise 37.4-3 *Sorting One-Dimensional Arrays While Preserving the Relationship with a Second Array*

Write a PHP script that lets the user enter the names of 20 lakes and their corresponding average area. The script should then sort them by average area in ascending order using the bubble sort algorithm.

Solution

In this exercise you need the following two arrays.



If you want to sort array `$areas` while preserving the relationship between the elements of the two arrays, you must rearrange the elements of the array `$names` as well. This means that every time two elements of the array `$areas` swap contents, the

corresponding elements of the array `$names` must swap contents as well. The PHP script is as follows.

file_37.4-3.php

```
<?php
define("LAKES", 20);

$names = array();
$areas = array();
for ($i = 0; $i <= LAKES - 1; $i++) {
    $names[$i] = trim(fgets(STDIN));
    $areas[$i] = trim(fgets(STDIN));
}

for ($m = 1; $m <= LAKES - 1; $m++) {
    for ($n = LAKES - 1; $n >= $m; $n--) {
        if ($areas[$n] < $areas[$n - 1]) {
            $temp = $areas[$n];
            $areas[$n] = $areas[$n - 1];
            $areas[$n - 1] = $temp;

            $temp = $names[$n];
            $names[$n] = $names[$n - 1];
            $names[$n - 1] = $temp;
        }
    }
}

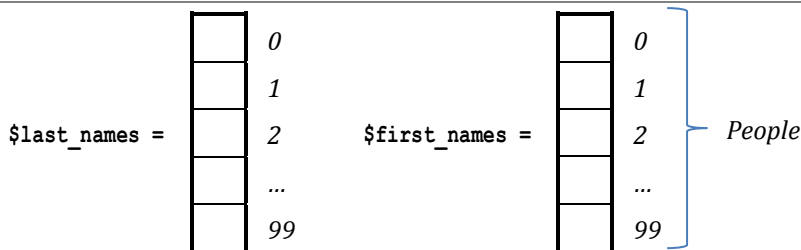
for ($i = 0; $i <= LAKES - 1; $i++) {
    echo $names[$i], "\t", $areas[$i], "\n";
}
?>
```

Exercise 37.4-4 *Sorting Last and First Names*

Write a PHP script that prompts the user to enter the last and first names of 100 people and then displays them with the last names sorted in alphabetical order. Moreover, if two or more people have the same last name, their first names should be displayed in alphabetical order as well.

Solution

In this exercise, you need the following two arrays.



Of course, you already know how to sort the array `$last_names` and at the same time preserve the relationship with the elements of the array `$first_names`. But, what you actually don't know here is how to handle the case when two last names are equal. According to the wording of the exercise, the corresponding first names should be also sorted alphabetically.

For your convenience, the basic version of the bubble sort algorithm is presented once again. Please note that this algorithm preserves the relationship between the elements of arrays `$last_names` and `$first_names`.

```
for ($m = 1; $m <= PEOPLE - 1; $m++) {
    for ($n = PEOPLE - 1; $n >= $m; $n--) {
        if ($last_names[$n] < $last_names[$n - 1]) {
            $temp = $last_names[$n];
            $last_names[$n] = $last_names[$n - 1];
            $last_names[$n - 1] = $temp;

            $temp = $first_names[$n];
            $first_names[$n] = $first_names[$n - 1];
            $first_names[$n - 1] = $temp;
        }
    }
}
```

To solve this exercise, however, this bubble sort algorithm must be adapted accordingly. According to this basic version of the bubble sort algorithm, when the last name at position `$n` is “less” than the last name at position `$n - 1`, the algorithm swaps the corresponding contents as usual. However, when the last name at position `$n` is equal to the last name at position `$n - 1`, the algorithm then must check if the corresponding first names are in the proper order. If not, they should swap their contents. The adapted bubble sort algorithm is shown in the next code fragment.

```
for ($m = 1; $m <= PEOPLE - 1; $m++) {
    for ($n = PEOPLE - 1; $n >= $m; $n--) {
        if ($last_names[$n] < $last_names[$n - 1]) {
            $temp = $last_names[$n];
            $last_names[$n] = $last_names[$n - 1];
            $last_names[$n - 1] = $temp;
```

```

    $temp = $first_names[$n];
    $first_names[$n] = $first_names[$n - 1];
    $first_names[$n - 1] = $temp;
}
elseif ($last_names[$n] == $last_names[$n - 1]) {
    if ($first_names[$n] < $first_names[$n - 1]) {
        $temp = $first_names[$n];
        $first_names[$n] = $first_names[$n - 1];
        $first_names[$n - 1] = $temp;
    }
}
}
}
}

```

The final PHP script is presented next.

file_37.4-4.php

<?php

```

define("PEOPLE", 100);

//Read arrays $first_names and $last_names
$first_names = array();
$last_names = array();
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    echo "Enter first name for person No. ", $i + 1, ": ";
    $first_names[$i] = trim(fgets(STDIN));
    echo "Enter last name for person No. ", $i + 1, ": ";
    $last_names[$i] = trim(fgets(STDIN));
}

for ($m = 1; $m <= PEOPLE - 1; $m++) {
    for ($n = PEOPLE - 1; $n >= $m; $n--) {
        if ($last_names[$n] < $last_names[$n - 1]) {
            $temp = $last_names[$n];
            $last_names[$n] = $last_names[$n - 1];
            $last_names[$n - 1] = $temp;

            $temp = $first_names[$n];
            $first_names[$n] = $first_names[$n - 1];
            $first_names[$n - 1] = $temp;
        }
    }
}

```

```

    }
elseif ($last_names[$n] == $last_names[$n - 1]) {
    if ($first_names[$n] < $first_names[$n - 1]) {
        $temp = $first_names[$n];
        $first_names[$n] = $first_names[$n - 1];
        $first_names[$n - 1] = $temp;
    }
}
}
}

//Display arrays $last_names and $first_names
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    echo $last_names[$i], "\t", $first_names[$i], "\n";
}
?>

```

Exercise 37.4-5 *Sorting a Two-Dimensional Array*

Write a code fragment that sorts each column of a two-dimensional array in ascending order.

Solution

An example of a two-dimension array is as follows.

	0	1	2	3	4	5	6
0							
1							
\$a = 2							
3							
4							

Since this array has seven columns, the bubble sort algorithm needs to be executed seven times, one for each column. Therefore, the whole bubble sort algorithm should be nested within a counted loop structure that iterates seven times.

But let's get things in the right order. Using the "from inner to outer" method, the next code fragment sorts only the first column (column index 0) of the two-dimensional array \$a. Assume variable \$j contains the value 0.

```

for ($m = 1; $m <= ROWS - 1; $m++) {
    for ($n = ROWS - 1; $n >= $m; $n--) {
        if ($a[$n][$j] < $a[$n - 1][$j]) {

```

```

        $temp = $a[$n][$j];
        $a[$n][$j] = $a[$n - 1][$j];
        $a[$n - 1][$j] = $temp;
    }
}
}

```

Now, in order to sort all columns, you can nest this code fragment in a counted loop structure that iterates for all of them, as follows.

```

for ($j = 0; $j <= COLUMNS - 1; $j++) {
    for ($m = 1; $m <= ROWS - 1; $m++) {
        for ($n = ROWS - 1; $n >= $m; $n--) {
            if ($a[$n][$j] < $a[$n - 1][$j]) {
                $temp = $a[$n][$j];
                $a[$n][$j] = $a[$n - 1][$j];
                $a[$n - 1][$j] = $temp;
            }
        }
    }
}
}

```

That wasn't so difficult, was it?

Exercise 37.4-6 *The Modified Bubble Sort Algorithm – Sorting One-Dimensional Arrays*

Write a PHP script that lets the user enter the weights of 30 people and then displays the three heaviest weights and the three lightest weights. Use the **modified** bubble sort algorithm.

Solution

To solve this exercise, the PHP script can sort the given data in ascending order and then display the elements at index positions 27, 28, and 29 (for the three heaviest weights) and the elements at index positions 0, 1 and 2 (for the three lightest weights).

But how does the modified version of the bubble sort algorithm actually work? Suppose you have the following array containing the weights of six people.

	0	1	2	3	4	5
\$w =	165	170	180	190	182	200

If you look closer, you can confirm for yourself that the only elements that are not in proper position are those at index positions 3 and 4. If you swap their values, the

array `$w` immediately becomes sorted! However, the bubble sort algorithm doesn't operate this way. Unfortunately, for this given array of six elements, it will perform five passes either way, with a total of $\frac{N(N-1)}{2} = 15$ compares, where N is the total number of array elements. For bigger arrays, the total number of compares that the bubble sort algorithm performs increases exponentially!

Of course the modified bubble sort algorithm can overcome this situation as follows: if a complete pass is performed and no swaps have been made, then the array is sorted and there is no need for further passes. To accomplish this, the PHP script can use a flag variable that indicates if any swaps were made. At the beginning of a pass, a value of `false` can be assigned to the flag variable; when a swap is made, a value of `true` is assigned. If, at the end of the pass, the flag is still `false`, this indicates that no swaps have been made, thus iterations should stop. The modified bubble sort is shown next in two versions.

The next approach uses the `break` statement and a flag variable (`swaps`).

```
for ($m = 1; $m <= ELEMENTS - 1; $m++) {
    $swaps = false;
    for ($n = ELEMENTS - 1; $n >= $m; $n--) {
        if ($w[$n] < $w[$n - 1]) {
            $temp = $w[$n];
            $w[$n] = $w[$n - 1];
            $w[$n - 1] = $temp;

            $swaps = true;
        }
    }
    if ($swaps == false) break;
}
```

The next approach, on the other hand, uses a pre-test loop structure and a flag variable.

```
$m = 1;
$swaps = true;
while ($m <= ELEMENTS - 1 && $swaps == true) {
    $swaps = false;
    for ($n = ELEMENTS - 1; $n >= $m; $n--) {
        if ($w[$n] < $w[$n - 1]) {
            $temp = $w[$n];
            $w[$n] = $w[$n - 1];
            $w[$n - 1] = $temp;

            $swaps = true;
        }
    }
}
```

```

    }
}
$m++;
}

```

So, let's now focus on the given exercise. The final PHP script is shown next. It uses the approach with the `break` statement, but you can easily rewrite it and make use of the second approach.

file_37.4-6.php

```

<?php
define("ELEMENTS", 20);

$w = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $w[$i] = trim(fgets(STDIN));
}

for ($m = 1; $m <= ELEMENTS - 1; $m++) {
    $swaps = false;
    for ($n = ELEMENTS - 1; $n >= $m; $n--) {
        if ($w[$n] < $w[$n - 1]) {
            $temp = $w[$n];
            $w[$n] = $w[$n - 1];
            $w[$n - 1] = $temp;

            $swaps = true;
        }
    }
    if ($swaps == false) break;
}

echo "The three heaviest weights are: \n";
echo $w[ELEMENTS - 3], "\t", $w[ELEMENTS - 2], "\t", $w[ELEMENTS - 1], "\n";
echo "The three lightest weights are: \n";
echo $w[0], "\t", $w[1], "\t", $w[2], "\t";

?>

```

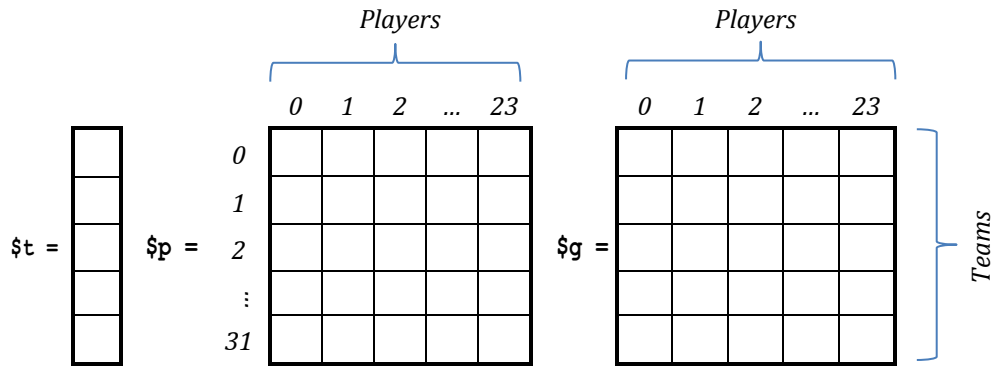
Exercise 37.4-7 *The Five Best Scorers*

Write a PHP script that prompts the user to enter the names of the 32 national teams of the FIFA World Cup, the names of the 24 players for each team, and the total number of

goals each player scored. The script should then display the name of each team along with its five best scorers. Use the modified bubble sort algorithm.

Solution

In this exercise you need the following three arrays.



Notice: To save paper short array names are used, but it is more or less obvious that array `$t` holds the names of the 32 national teams, array `$p` holds the names of the 24 players of each team, and array `$g` holds the total number of goals each player scored.

Since you need to find the five best scorers for each team, the PHP script must sort each row of array `$g` in descending order but it should also take care to preserve the relationship with the elements of arrays `$p`. This means that, every time the bubble sort algorithm swaps the contents of two elements of array `$g`, the corresponding elements of array `$p` should be swapped as well. When sorting is completed, the five best scorers should appear in the first five columns.

The “from inner to outer” method is used again. The following code fragment sorts the first row (row index 0) of array `$g` in descending order and, at the same time, takes care to preserve the relationship with elements of array `$p`. Assume variable `$i` contains the value 0.

```
$m = 1;
$swaps = true;
while ($m <= PLAYERS - 1 && $swaps == true) {
    $swaps = false;
    for ($n = PLAYERS - 1; $n >= $m; $n--) {
        if ($g[$i][$n] < $g[$i][$n - 1]) {
            $temp = $g[$i][$n];
            $g[$i][$n] = $g[$i][$n - 1];
            $g[$i][$n - 1] = $temp;
        }
    }
}
```

```

    $temp = $p[$i][$n];
    $p[$i][$n] = $p[$i][$n - 1];
    $p[$i][$n - 1] = $temp;

    $swaps = true;
}
}
$m++;
}

```

Now, in order to sort all rows, you need to nest this code fragment in a counted loop structure that iterates for all of them, as shown next.

```

for ($i = 0; $i <= TEAMS - 1; $i++) {
    $m = 1;
    $swaps = true;
    while ($m <= PLAYERS - 1 && $swaps == true) {
        $swaps = false;
        for ($n = PLAYERS - 1; $n >= $m; $n--) {
            if ($g[$i][$n] < $g[$i][$n - 1]) {
                $temp = $g[$i][$n];
                $g[$i][$n] = $g[$i][$n - 1];
                $g[$i][$n - 1] = $temp;

                $temp = $p[$i][$n];
                $p[$i][$n] = $p[$i][$n - 1];
                $p[$i][$n - 1] = $temp;

                $swaps = true;
            }
        }
        $m++;
    }
}

```

The final PHP script is as follows.

file_37.4-7.php

```
<?php
```

```

define("TEAMS", 32);
define("PLAYERS", 24);

```

```
//Read team names, player names and goals all together
```

```

$t = array();
$p = array(array());
$g = array(array());
for ($i = 0; $i <= TEAMS - 1; $i++) {
    echo "Enter name for team No. ", $i + 1, ": ";
    $t[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= PLAYERS - 1; $j++) {
        echo "Enter name of player No. ", $j + 1, ": ";
        $p[$i][$j] = trim(fgets(STDIN));
        echo "Enter goals of player No. ", $j + 1, ": ";
        $g[$i][$j] = trim(fgets(STDIN));
    }
}

//sort array $g
for ($i = 0; $i <= TEAMS - 1; $i++) {
    $m = 1;
    $swaps = true;
    while ($m <= PLAYERS - 1 && $swaps == true) {
        $swaps = false;
        for ($n = PLAYERS - 1; $n >= $m; $n--) {
            if ($g[$i][$n] > $g[$i][$n - 1]) {
                $temp = $g[$i][$n];
                $g[$i][$n] = $g[$i][$n - 1];
                $g[$i][$n - 1] = $temp;

                $temp = $p[$i][$n];
                $p[$i][$n] = $p[$i][$n - 1];
                $p[$i][$n - 1] = $temp;

                $swaps = true;
            }
        }
        $m++;
    }
}

//Display 5 best scorers of each team
for ($i = 0; $i <= TEAMS - 1; $i++) {
    echo "\nBest scorers of ", $t[$i], "\n";
}

```

```

echo "-----\n";
for ($j = 0; $j <= 4; $j++) {
    echo $p[$i][$j], " scored ", $g[$i][$j], " goals\n";
}
}
?>

```

Exercise 37.4-8 *The Selection Sort Algorithm – Sorting One-Dimensional Arrays*

Write a code fragment that sorts the elements of an array in ascending order using the selection sort algorithm.

Solution

The selection sort algorithm is inefficient for large scale data, as is the bubble sort algorithm, but it generally performs better than the latter. It is the simplest of all the sorting algorithms and performs well on computer systems in which limited main memory (RAM) comes into play.

The algorithm finds the smallest (or largest, depending on sorting order) element of the array and swaps its content with that at position 0. Then the process is repeated for the remainder of the array; the next smallest (or largest) element is found and put into the next position, until all elements are examined.

For example, let's try to sort the following array in ascending order.

	0	1	2	3	4	5
A =	18	19	39	36	4	9

The lowest value is the value 4 at position 4. According to the selection sort algorithm, this element swaps its content with that at position 0. The array **A** becomes

	0	1	2	3	4	5
A =	4	19	41	36	18	9

The lowest value in the remainder of the array is the value 9 at position 5. This element swaps its content with that at position 1. The array **A** becomes

	0	1	2	3	4	5
A =	4	9	41	36	18	19

The lowest value in the remainder of the array is the value 18 at position 4. This element swaps its content with that at position 2. The array **A** becomes

	0	1	2	3	4	5
A =	4	9	18	36	41	19

Proceeding the same way, the next lowest value is the value 19 at position 5. The array **A** becomes

	0	1	2	3	4	5
A =	4	9	18	19	41	36

Finally, the next lowest value is the value 36 at position 5. The array **A** becomes finally sorted in ascending order!

	0	1	2	3	4	5
A =	4	9	18	19	36	41

Now, let's write the corresponding PHP script. The "from inner to outer" method is used in order to help you better understand the whole process. The next code fragment finds the smallest element and then swaps its content with that at position 0. Please note that this is the inner (nested) loop control structure. Assume variable `$m` contains the value 0.

```
$min = $a[$m];
$index_of_min = $m;
for ($n = $m; $n <= ELEMENTS - 1; $n++) {
    if ($a[$n] < $min) {
        $min = $a[$n];
        $index_of_min = $n;
    }
}
//Minimum found! Now, swap values.
$temp = $a[$m];
$a[$m] = $a[$index_of_min];
$a[$index_of_min] = $temp;
```

Now, in order to repeat the process for all elements of the array, you can nest this code fragment within a counted loop structure that iterates for all elements.

The final selection sort algorithm that sorts an array in ascending order is as follows.

```
for ($m = 0; $m <= ELEMENTS - 1; $m++) {
    $min = $a[$m];
    $index_of_min = $m;
    for ($n = $m; $n <= ELEMENTS - 1; $n++) {
        if ($a[$n] < $min) {
            $min = $a[$n];
            $index_of_min = $n;
        }
    }
    //Minimum found! Now, swap values.
    $temp = $a[$m];
    $a[$m] = $a[$index_of_min];
    $a[$index_of_min] = $temp;
}
```

```
        $index_of_min = $n;
    }
}
//Swap values
$temp = $a[$m];
$a[$m] = $a[$index_of_min];
$a[$index_of_min] = $temp;
}
```

Notice: If you wish to sort an array in descending order, all you need to do is search for maximum instead of minimum values.

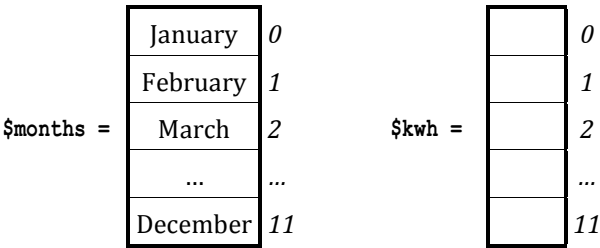
Remember! As in a bubble sort algorithm, in order to sort alphanumeric data, you can do one simple thing: leave the algorithm as is!

Exercise 37.4-9 Sorting One-Dimensional Arrays While Preserving the Relationship with a Second Array

Write a PHP script that prompts the user to enter the electric meter reading recorded at the end of each month for a period of one year. It then displays each reading (in descending order) along with the name of the corresponding month. Use the selection sort algorithm.

Solution

In this exercise you need the following two one-dimensional arrays.



The approach is the same as in the bubble sort algorithm. While the selection sort algorithm sorts the elements of array \$kwh, the relationship with the elements of array \$months must also be preserved. This means that every time two elements of array \$kwh swap contents, the corresponding elements of array \$months must swap their contents as well. The PHP script is as follows.

```
file_37.4-9.php
<?php
define("ELEMENTS", 12);
$months = array("January", "February", "March", "April",
```

```
        "May", "June", "July", "August",  
        "September", "October", "November", "December");  
  
$kwh = array();  
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    echo "Enter kWh for ", $months[$i], ": ";  
    $kwh[$i] = trim(fgets(STDIN));  
}  
  
for ($m = 0; $m <= ELEMENTS - 1; $m++) {  
    $max = $kwh[$m];  
    $index_of_max = $m;  
    for ($n = $m; $n <= ELEMENTS - 1; $n++) {  
        if ($kwh[$n] > $max) {  
            $max = $kwh[$n];  
            $index_of_max = $n;  
        }  
    }  
}  
  
//Swap values of $kwh  
$temp = $kwh[$m];  
$kwh[$m] = $kwh[$index_of_max];  
$kwh[$index_of_max] = $temp;  
  
//Swap values of $months  
$temp = $months[$m];  
$months[$m] = $months[$index_of_max];  
$months[$index_of_max] = $temp;  
}  
  
for ($i = 0; $i <= ELEMENTS - 1; $i++) {  
    echo $months[$i], ": ", $kwh[$i], "\n";  
}  
?>
```

Exercise 37.4-10 *The Insertion Sort Algorithm – Sorting One-Dimensional Arrays*

Write a code fragment that sorts the elements of an array in ascending order using the insertion sort algorithm.

Solution

The insertion sort algorithm is inefficient for large scale data, as are the selection and the bubble sort algorithms, but it generally performs better than either of them. Moreover, the insertion sort algorithm can prove very fast when sorting very small arrays— even faster than the quicksort algorithm.

The insertion sort algorithm resembles the way you might sort playing cards. You start with all the cards face down on the table. The cards represent the unsorted “array.” In the beginning your left hand is empty, but in the end this hand will hold the sorted cards. The process goes as follows: you remove from the table one card at a time and insert it into the correct position in your left hand. To find the correct position for a card, you compare it with each of the cards already in your hand, from right to left. At the end, there should be no cards on the table and your left hand will hold all the cards, sorted.

For example, let’s try to sort the following array in ascending order. To better understand this example, the first three elements of the array are already sorted for you.

	0	1	2	3	4	5	6
A =	3	15	24	8	10	18	9

The element at position 3 (which is 8) is removed from the array and all elements on its left with a value greater than 8 are shifted to the right. The array A becomes

	0	1	2	3	4	5	6
A =	3		15	24	10	18	9

Now that a position has been released, the value 8 is inserted in there. The array becomes

	0	1	2	3	4	5	6
A =	3	8	15	24	10	18	9

The element at position 4 (which is 10) is removed from the array and all elements on its left with a value greater than 10 are shifted to the right. The array A becomes

	0	1	2	3	4	5	6
A =	3	8		15	24	18	9

Now that a position has been released, the value of 10 is inserted in there. The array becomes

	0	1	2	3	4	5	6
A =	3	8	10	15	24	18	9

The element at position 5 is removed from the array and all elements on its left with a value greater than 18 are shifted to the right. The array A becomes

	0	1	2	3	4	5	6
A =	3	8	10	15		24	9

Now that a position has been released, the value of 18 is inserted in there. The array becomes

	0	1	2	3	4	5	6
A =	3	8	10	15	18	24	9

The element at position 6 is removed from the array and all elements on its left with a value greater than 9 are shifted to the right. The array A becomes

	0	1	2	3	4	5	6
A =	3	8		10	15	18	24

Now that a position has been released, the value of 9 is inserted in there. This is the last step of the process. The algorithm finishes and the array is now sorted.

	0	1	2	3	4	5	6
A =	3	8	9	10	15	18	24

Remember! What the algorithm actually does is to check the unsorted elements one by one and insert each one in the appropriate position among those considered already sorted.

The code fragment that sorts an array in ascending order using the insertion sort algorithm is as follows.

```
for ($m = 1; $m <= ELEMENTS - 1; $m++) {
    // "Remove" the element at index position $m from the array and
    // keep it in variable $element
    $element = $a[$m];

    // Shift appropriate elements to the right
    $n = $m;
    while ($n > 0 && $a[$n - 1] > $element) {
```

```
    $a[$n] = $a[$n - 1];  
    $n--;  
}  
  
//insert the previously "removed" element at index position $n  
$a[$n] = $element;  
}
```

Notice: Please note that the element at index position m is not actually removed from the array but is in fact overwritten when shifting to the right is performed. This is why its value is kept in variable $element$ before shifting the elements.

Notice: If you wish to sort an array in descending order, all you need to do is alter the Boolean expression of the `while` statement to $n > 0 \ \&\& \ a[n - 1] < element$.

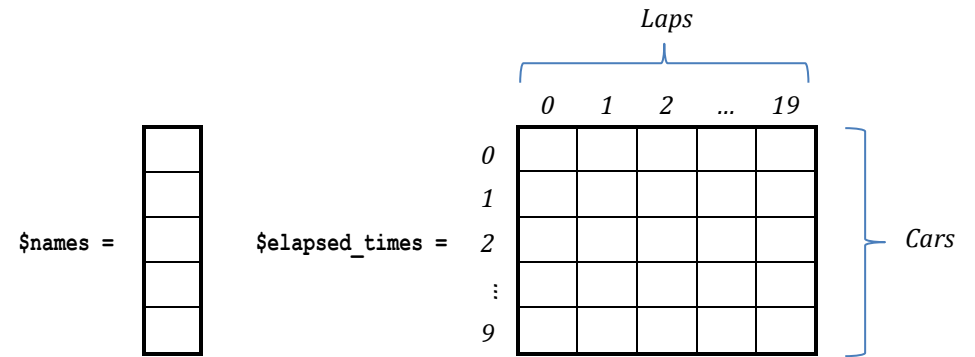
Remember! As in previous sort algorithms, in order to sort alphanumeric data, you don't have to change anything in the algorithm!

Exercise 37.4-11 *The Three Worst Elapsed Times*

Ten race car drivers run their cars as fast as possible on a racing track. Each car runs 20 laps and for each lap the corresponding elapsed time (in seconds) is recorded. Write a PHP script that prompts the user to enter the name of each driver and their elapsed time for each lap. The script should then display the name of each driver along with his or her three worst elapsed times. Use the insertion sort algorithm.

Solution

In this exercise, you need the following two arrays.



After the user enters all data, the PHP script must sort each row of the array in descending order but, in the end, should display only the first three columns.

Using the “from inner to outer” method, the next code fragment sorts only the first row (row index 0) of the two-dimensional array `$elapsed_times` in descending order using the insertion sort algorithm. Assume variable `$i` contains the value 0.

```
for ($m = 1; $m <= LAPS - 1; $m++) {
    $element = $elapsed_times[$i][$m];
    $n = $m;
    while($n > 0 && $elapsed_times[$i][$n - 1] < $element) {
        $elapsed_times[$i][$n] = $elapsed_times[$i][$n - 1];
        $n--;
    }
    $elapsed_times[$i][$n] = $element;
}
```

Now, in order to sort all rows, you need to nest this code fragment in a counted loop structure that iterates for all of them, as follows.

```
for ($i = 0; $i <= CARS - 1; $i++) {
    for ($m = 1; $m <= LAPS - 1; $m++) {
        $element = $elapsed_times[$i][$m];
        $n = $m;
        while($n > 0 && $elapsed_times[$i][$n - 1] < $element) {
            $elapsed_times[$i][$n] = $elapsed_times[$i][$n - 1];
            $n--;
        }
        $elapsed_times[$i][$n] = $element;
    }
}
```

And now, let's focus on the given exercise. The final PHP script is as follows.

file_37.4-11.php

```
<?php
define("CARS", 10);
define("LAPS", 20);

//Read names and elapsed times all together
$names = array();
$elapsed_times = array(array());
for ($i = 0; $i <= CARS - 1; $i++) {
    echo "Enter name for driver No. ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= LAPS - 1; $j++) {
        echo "Enter elapsed time for lap No. ", $j + 1, ": ";
```

```

        $elapsed_times[$i][$j] = trim(fgets(STDIN));
    }
}

for ($i = 0; $i <= CARS - 1; $i++) {
    for ($m = 1; $m <= LAPS - 1; $m++) {
        $element = $elapsed_times[$i][$m];
        $n = $m;
        while($n > 0 && $elapsed_times[$i][$n - 1] < $element) {
            $elapsed_times[$i][$n] = $elapsed_times[$i][$n - 1];
            $n--;
        }
        $elapsed_times[$i][$n] = $element;
    }
}

//Display 3 worst elapsed times
for ($i = 0; $i <= CARS - 1; $i++) {
    echo "\nWorst elapsed times of ", $names[$i], "\n";
    echo "-----\n";
    for ($j = 0; $j <= 2; $j++) {
        echo $elapsed_times[$i][$j], "\n";
    }
}

?>

```

37.5 Searching Elements in Arrays

In computer science, a search algorithm is an algorithm that searches for an item with specific properties within a set of data. In the case of an array, a search algorithm does exactly the same thing; it searches the array to find the element, or elements, that equal a given value.

There are two cases for searching elements in arrays.

- You want to search in an array for a given value and find the element (or its corresponding index) that is equal to that given value.
- You want to search in an array for a given value and find **all** the elements (or their corresponding indexes) that are equal to that given value.

The most commonly used search algorithms are:

- the linear (or sequential) search algorithm
- the binary search algorithm

Both linear and binary search algorithms have advantages and disadvantages.

Exercise 37.5-1 ***The Linear Search Algorithm – Searching in a One-Dimensional Array that may Contain the Same Value Multiple Times***

Write a code fragment that searches a one-dimensional array for a given value. Assume that the array may contain the same value multiple times. Use the linear search algorithm.

Solution

The linear (or sequential) search algorithm checks if the first element of the array is equal to a given value, then checks the second element, then the third, and so on until the end of the array. Since this process of checking elements one by one is quite slow, the linear search algorithm is suitable for arrays with few elements.

The code fragment is shown next. It looks for a given value `$needle` in the array `$haystack`!

```
echo "Enter a value to search: ";
$needle = trim(fgets(STDIN));

$found = false;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($haystack[$i] == $needle) {
        echo $needle, " found at position: ", $i, "\n";
        $found = true;
    }
}

if ($found == false) {
    echo "Nothing found!";
}
```

Exercise 37.5-2 ***Display the Last Names of All Those People Who Have the Same First Name***

Write a PHP script that prompts the user to enter the names of 20 people: their first names in the array `$first_names`, and their last names in the array `$last_names`. The script should then ask the user for a first name, upon which it will search and display the last names of all those whose first name equals the given one.

Solution

The solution is as follows.

file_37.5-2.php

```
<?php
define("PEOPLE", 20);

$first_names = array();
$last_names = array();
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    echo "Enter first name: ";
    $first_names[$i] = trim(fgets(STDIN));
    echo "Enter last name: ";
    $last_names[$i] = trim(fgets(STDIN));
}

echo "Enter a first name to search: ";
$needle = trim(fgets(STDIN));

$found = false;
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    if (strtoupper($first_names[$i]) == strtoupper($needle)) {
        echo $last_names[$i], "\n";
        $found = true;
    }
}

if ($found == false) {
    echo "No one found!";
}

?>
```

Notice: Even though it is not very clear in the wording of the exercise, it is true that the array `$first_names` may contain a value multiple times. How rare is it to meet two people named “John,” for example?

Notice: Since the script works with alphanumeric data, the `strtoupper()` function is required so that the script can operate correctly for any given value. For example, if the value “John” exists in the array `$first_names` and the user wants to search for the value “JOHN,” the `strtoupper()` function ensures that the script finds all Johns.

Exercise 37.5-3 *Searching in a One-Dimensional Array that Contains Unique Values*

Write a code fragment that searches a one-dimensional array for a given value. Assume that each value in the array is unique. Use the linear search algorithm.

Solution

This case is quite different from the previous ones. Since each value in the array is unique, when the given value is found, there is no need to iterate without reason until the end of the array, thus wasting CPU time. There are three approaches, actually! Let's analyze them all!

First Approach – Using the break statement

In this approach, when the given value is found, a `break` statement is used to break out of the counted loop structure. The solution is as follows.

```
echo "Enter a value to search: ";
$needle = trim(fgets(STDIN));

$found = false;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($haystack[$i] == $needle) {
        echo $needle, " found at position: ", $i, "\n";
        $found = true;
        break;
    }
}

if ($found == false) {
    echo "Nothing found!";
}
```

Or you can do the same, in a little bit different way.

```
echo "Enter a value to search: ";
$needle = trim(fgets(STDIN));

$index_position = -1;
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    if ($haystack[$i] == $needle) {
        $index_position = $i;
        break;
    }
}
```

```
if ($index_position == -1) {  
    echo "Nothing found!";  
}  
else {  
    echo $needle, " found at position: ", $index_position;  
}
```

Second Approach – Using a flag

This book tries to use the `break` statement as little as possible, and only in certain situations in which it really helps you make your code better. That's because the `break` statement doesn't actually exist in all computer languages; and since this book's intent is to teach you "Algorithmic Thinking" (and not just special statements that only PHP supports), let's look at an alternate approach.

To break out of a loop using a flag, there is one methodology that you can always follow: the counted loop structure must be converted to a pre-test loop structure, and the Boolean expression of the loop must be modified to evaluate an extra variable. In the next code fragment this extra variable is the variable `$found` which, when the given value is found within array `$haystack`, forces the flow of execution to immediately exit the loop.

```
echo "Enter a value to search: ";  
$needle = trim(fgets(STDIN));  
  
$found = false;  
$i = 0;  
while ($i <= ELEMENTS - 1 && $found == false) {  
    if ($haystack[$i] == $needle) {  
        $found = true;  
        $index_position = $i;  
    }  
    $i++;  
}  
  
if ($found == false) {  
    echo "Nothing found!";  
}  
else {  
    echo $needle, " found at position: ", $index_position;  
}
```


Third Approach – Using only a pre-test loop structure

This approach is probably the most efficient one. Also, it is so easy to understand the way it works that there is no need to explain anything. The code fragment is as follows.

```
echo "Enter a value to search: ";
$needle = trim(fgets(STDIN));

$i = 0;
while ($i < ELEMENTS - 1 && $haystack[$i] != $needle) {
    $i++;
}

if ($haystack[$i] != $needle) {
    echo "Nothing found!";
}
else {
    echo $needle, " found at position: ", $i;
}
```

Exercise 37.5-4 Searching for a Given Social Security Number

In the United States, the Social Security Number (SSN) is a nine-digit identity number applied to all U.S. citizens in order to identify them for the purposes of Social Security. Write a PHP script that prompts the user to enter the SSN and the first and last names of 100 people. The script should then ask the user for an SSN, upon which it will search and display the first and last name of the person who holds that SSN.

Solution

According to everything you learned so far, the solution to this exercise is as follows.

file_37.5-4.php

```
<?php
define("PEOPLE", 100);

$SSNs = array();
$first_names = array();
$last_names = array();
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    echo "Enter SSN: ";
    $SSNs[$i] = trim(fgets(STDIN));
    echo "Enter first name: ";
    $first_names[$i] = trim(fgets(STDIN));
```

```
    echo "Enter last name: ";
    $last_names[$i] = trim(fgets(STDIN));
}

echo "Enter an SSN to search: ";
$needle = trim(fgets(STDIN));

$i = 0;
while ($i < PEOPLE - 1 && $SSNs[$i] != $needle) {
    $i++;
}

if ($SSNs[$i] != $needle) {
    echo "Nothing found!";
}
else {
    echo $first_names[$i], " ", $last_names[$i];
}
?>
```

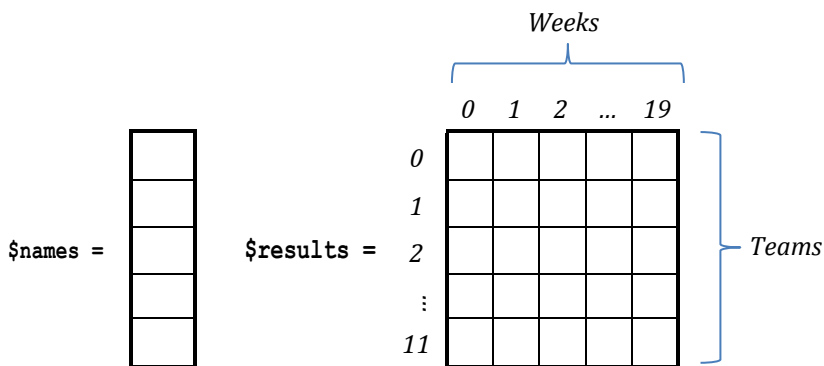
Notice: In the United States, there is no possibility that two or more people will have the same SSN. Thus, even though it is not very clear in the wording of the exercise, each value in array `$SSNs` is unique!

Exercise 37.5-5 *Searching in a Two-Dimensional Array that may Contain the Same Value Multiple Times*

Twelve teams participate in a football tournament, and each team plays 20 games, one game each week. Write a PHP script that prompts the user to enter the name of each team and the letter “W” for win, “L” for loss, or “T” for tie (draw) for each game. Then the script should prompt the user for a letter (W, L, or T) and display, for each team, the week number(s) in which the team won, lost, or tied respectively. For example, if the user enters “L,” the PHP script should search and display, for each team, the week numbers (e.g., week 3, week 14, and so on) in which the team lost the game.

Solution

In this exercise, you need the following two arrays.



The corresponding PHP script is as follows.

file_37.5-5.php

```
<?php
define("TEAMS", 20);
define("WEEKS", 12);

$names = array();
$results = array(array());
for ($i = 0; $i <= TEAMS - 1; $i++) {
    echo "Enter name for team No. ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= WEEKS - 1; $j++) {
        echo "Enter result for week No. ", $j + 1, " for ", $names[$i], ": ";
        $results[$i][$j] = trim(fgets(STDIN));
    }
}

echo "Enter a result to search: ";
$needle = trim(fgets(STDIN));

for ($i = 0; $i <= TEAMS - 1; $i++) {
    $found = false;
    echo "Found results for ", $names[$i], "\n";
    for ($j = 0; $j <= WEEKS - 1; $j++) {
        if (strtoupper($results[$i][$j]) == strtoupper($needle)) {
            echo "Week No ", $j + 1, "\n";
            $found = true;
        }
    }
}
```

```

if ($found == false) {
    echo "No results!\n";
}
}
?>

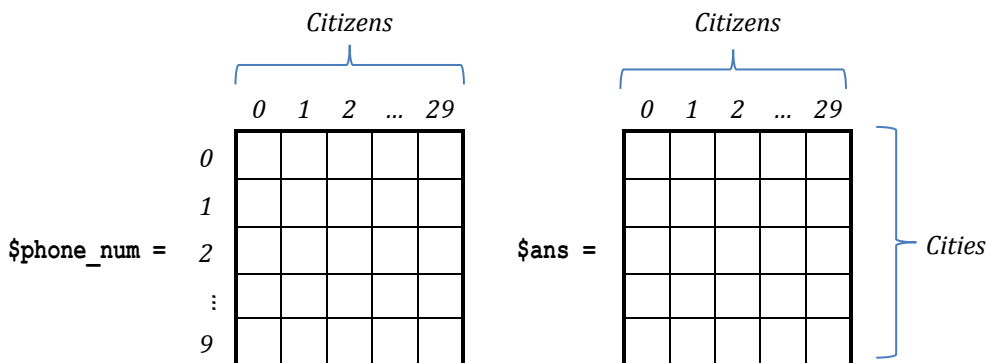
```

Exercise 37.5-6 *Searching in a Two-Dimensional Array that Contains Unique Values*

A public opinion polling company makes phone calls in 10 cities and asks 30 citizens in each city whether or not they exercise. Write a PHP script that prompts the user to enter each citizen's phone number and their answer (Y for Yes, N for No, S for Sometimes). The script should then prompt the user to enter a phone number, and it will search and display the answer that was given at this phone number. The script should also validate data input and accept only the values Y, N, or S as an answer.

Solution

In this exercise, you need the following two arrays.



The corresponding PHP script is as follows.

file_37.5-6.php

```

<?php
define("CITIES", 10);
define("CITIZENS", 30);

$phone_num = array(array());
$ans = array(array());
for ($i = 0; $i <= CITIES - 1; $i++) {
    echo "City No. ", $i + 1, "\n";
    for ($j = 0; $j <= CITIZENS - 1; $j++) {

```

```
echo "Enter phone number of citizen No. ", $j + 1, ": ";
$phone_num[$i][$j] = trim(fgets(STDIN));

echo "Enter the answer of citizen No. ", $j + 1, ": ";
$ans[$i][$j] = strtoupper(trim(fgets(STDIN)));
while ($ans[$i][$j] != "Y" &&
       $ans[$i][$j] != "N" &&
       $ans[$i][$j] != "S") {
    echo "Wrong answer. Please enter a valid answer: ";
    $ans[$i][$j] = strtoupper(trim(fgets(STDIN)));
}
}
}

echo "Enter a phone number to search: ";
$needle = trim(fgets(STDIN));

$found = false;
$i = 0;
while ($i <= CITIES - 1 && $found == false) {
    $j = 0;
    while ($j <= CITIZENS - 1 && $found == false) {
        if ($phone_num[$i][$j] == $needle) {
            $found = true;
            $position_i = $i;
            $position_j = $j;
        }
        $j++;
    }
    $i++;
}

if ($found == false) {
    echo "Phone number not found!";
}
else {
    echo "Phone number ", $phone_num[$position_i][$position_j] , " gave ";

    switch ($ans[$position_i][$position_j]) {
        case "Y":
```

```
        echo "Yes";
        break;
    case "N":
        echo "No";
        break;
    default:
        echo "Sometimes";
    }

    echo "' as an answer";
}

?>
```

Exercise 37.5-7 *Checking if a Value Exists in all Columns*

Write a PHP script that lets the user enter numeric values into a 20×30 array. After all of the values have been entered, the script then lets the user enter a value. A message should be displayed if the given value exists, at least once, in each column of the array.

Solution

This exercise can be solved using the linear search algorithm and a counter variable `$count`. The PHP script can iterate through the first column and if the given value is found, the variable `$count` should increment by one. Then the script can iterate through the second column and if the given value is found again, the variable `$count` should again increment by one. This process should repeat until all columns have been examined. At the end of the process, if the value of `$count` is equal to the total number of columns, this means that the given value exists, at least once, in each column of the array.

Let's use the "from inner to outer" method. The following code fragment searches in first column (column index 0) of the array and if the given value is found, variable `$count` increments by one. Assume variable `$j` contains the value 0.

```
$found = false;
$i = 0;
while ($i <= ROWS - 1 && $found == false) {
    if ($haystack[$i][$j] == $needle) {
        $found = true;
    }
    $i++;
}

if ($found == true) {
    $count++;
}
```

```
}
```

Now you can nest this code fragment in a counted loop structure that iterates for all columns.

```
for ($j = 0; $j <= COLUMNS - 1; $j++) {  
    $found = false;  
    $i = 0;  
    while ($i <= ROWS - 1 && $found == false) {  
        if ($haystack[$i][$j] == $needle) {  
            $found = true;  
        }  
        $i++;  
    }  
  
    if ($found == true) {  
        $count++;  
    }  
}
```

You are almost ready—but think about it first! If the inner pre-test loop structure doesn't find the given value in a column, the outer loop control structure should stop iterating. It is pointless to continue because the given value does not exist in at least one column. Thus, a better approach would be to use a post-test loop structure for the outer loop as shown next.

```
$j = 0;  
do {  
    $found = false;  
    $i = 0;  
    while ($i <= ROWS - 1 && $found == false) {  
        if ($haystack[$i][$j] == $needle) {  
            $found = true;  
        }  
        $i++;  
    }  
  
    if ($found == true) {  
        $count++;  
    }  
  
    $j++;  
} while ($j <= COLUMNS - 1 && $found == true);
```

The main idea is to break out of the loop when the given value is **not** found in one column.

Notice: You can use a pre-test loop structure as well, but you need to initially assign the value `true` to the variable `$found`.

The final PHP script is as follows.

file_37.5-7.php

```
<?php
define("ROWS", 20);
define("COLUMNS", 30);

$haystack = array(array());
for ($i = 0; $i <= ROWS - 1; $i++) {
    for ($j = 0; $j <= COLUMNS - 1; $j++) {
        $haystack[$i][$j] = trim(fgets(STDIN));
    }
}

echo "Enter a value to search: ";
$needle = trim(fgets(STDIN));

$count = 0;
$j = 0;
do {
    $found = false;
    $i = 0;
    while ($i <= ROWS - 1 && $found == false) {
        if ($haystack[$i][$j] == $needle) {
            $found = true;
        }
        $i++;
    }

    if ($found == true) {
        $count++;
    }

    $j++;
} while ($j <= COLUMNS - 1 && $found == true);
```



```
if ($count == COLUMNS) {
    echo $needle, " found in every column!";
}
?>
```

Notice: If you need a message to be displayed when a given value exists at least once in each **row**, the PHP script can do the same as previously shown; however, instead of iterating through columns, it has to iterate through rows.

Exercise 37.5-8 *The Binary Search Algorithm – Searching in a Sorted One-Dimensional Array*

Write a code fragment that searches a sorted one-dimensional array for a given value. Use the binary search algorithm.

Solution

The binary search algorithm is considered very fast and can be used with large scale data. Its main disadvantage, though, is that the data need to be sorted.

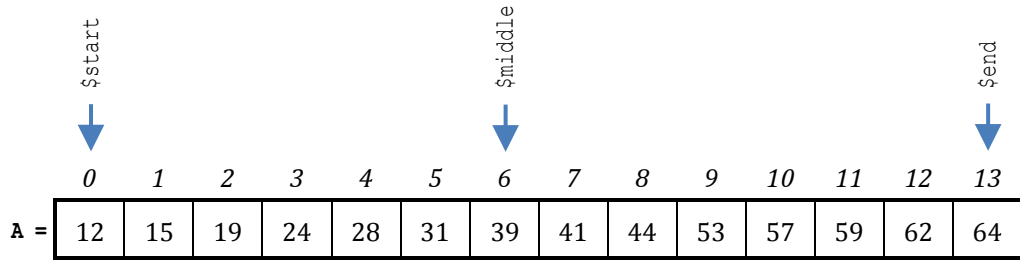
The main idea of the binary search algorithm is to check the value of the element in the middle (approximately). If it is not the “needle in the haystack” that you are looking for, and what you are looking for is actually smaller than the value of the middle element, it means that the “needle” must be in the first half of the array. On the other hand, if what you are looking for is larger than the value of the middle element, it means that the “needle” must be in the last half of the array.

The binary search algorithm continues by checking the value of the middle element in the remaining half part of the array. If it’s still not the “needle,” the search narrows to the half of the remaining part of the array that the “needle” could be in. This “splitting” process continues until the “needle” is found or the remaining part of the array consists of only one element. If that element is not the “needle,” then the given value is not in the array.

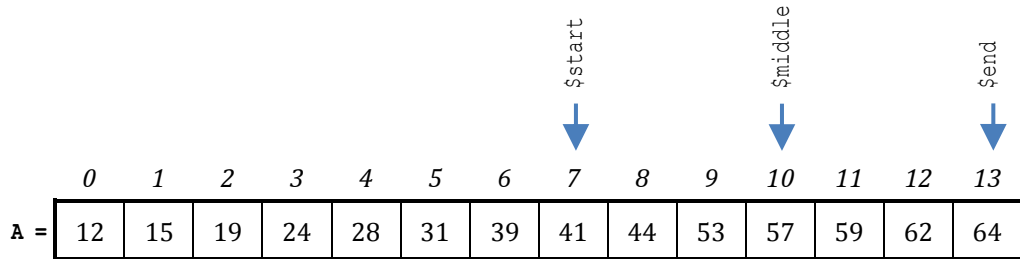
Confused? Let’s try to analyze the binary search algorithm through an example. The following array contains numeric values in ascending order. Assume that the “needle” that you are looking for is the value 44.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A =	12	15	19	24	28	31	39	41	44	53	57	59	62	64

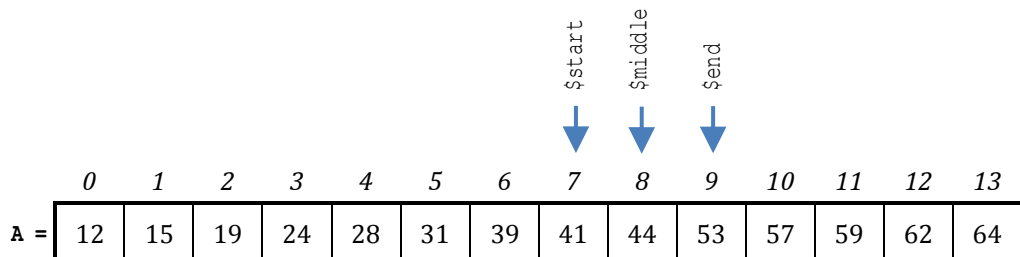
Three variables are used. Initially, variable `$start` contains the value 0 (this is the index of the first element), variable `$end` contains the value 12 (this is the index of the last element) and variable `$middle` contains the value 6 (this is approximately the index of the middle element).



The “needle” (value 44) that you are looking for is larger than the value of 39 in the middle, thus the element that you are looking for must be in the last half of the array. Therefore, variable `$start` is updated to point to index position 7 and variable `$middle` is updated to a point in the middle between `$start` (the new one) and `$end`.



Now, the “needle” (value 44) that you are looking for is smaller than the value of 57 in the middle, thus the element that you are looking for must be in the first half of the remaining part of the array. Therefore, it is the variable `$end` that is now updated to point to index position 9, and variable `$middle` is updated to point to the middle between `$start` and `$end` (the new one).



You are done! The “needle” has been found at index position 8 and the whole process can stop!

Remember! Each unsuccessful comparison reduces the number of elements left to check by half!

Now, let's see the corresponding code fragment.

```
$left = 0;
$right = ELEMENTS - 1;
$found = false;
while ($left <= $right && $found == false) {
    $middle = intval(($left + $right) / 2);

    if ($haystack[$middle] > $needle) {
        $right = $middle - 1;
    }
    elseif ($haystack[$middle] < $needle) {
        $left = $middle + 1;
    }
    else {
        $found = true;
        $index_position = $middle;
    }
}

if ($found == false) {
    echo "Nothing found!";
}
else {
    echo $needle, " found at position: ", $index_position;
}
```

Notice: Using the binary search algorithm on the array of the example, the value of 44 can be found within just three iterations. On the other hand, for the same data, the linear search algorithm would need nine iterations!

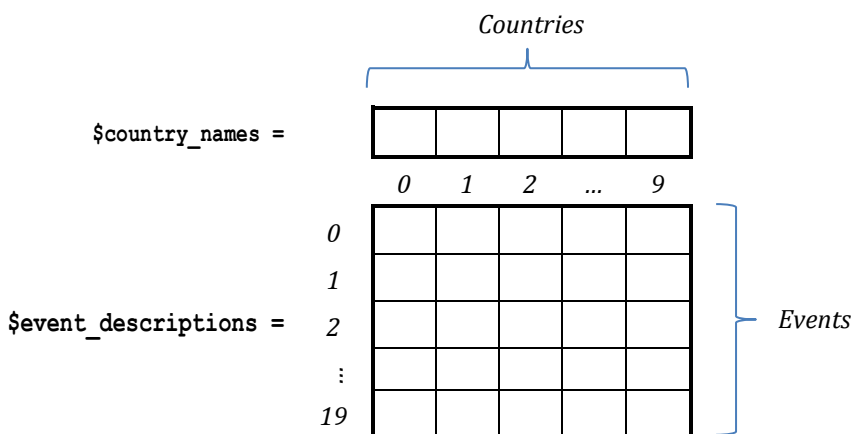
Notice: If the array contains a value multiple times, the binary search algorithm can find only one occurrence.

Exercise 37.5-9 *Display all the Historical Events for a Country*

Write a PHP script that prompts the user to enter the names of 10 countries and 20 important historical events for each country, including a brief description of each event. The PHP script should then prompt the user to enter a country, and it will search and display all events for that country. Use the binary search algorithm. Assume that the user enters the names of the countries alphabetically.

Solution

In this exercise, the following two arrays are required.



Assume that the user enters a country to search for, and the binary search algorithm finds that country, for example, at index position 2 of array `$country_names`. The script can then use this value of 2 as a column index for the array `$event_descriptions`, and it will display all the event descriptions of column 2.

The PHP script is as follows.

file_37.5-9.php

```
<?php
define("EVENTS", 20);
define("COUNTRIES", 10);

$country_names = array();
$event_descriptions = array(array());
for ($j = 0; $j <= COUNTRIES - 1; $j++) {
    echo "Enter Country No. ", $j + 1, ": ";
    $country_names[$j] = trim(fgets(STDIN));
    for ($i = 0; $i <= EVENTS - 1; $i++) {
        echo "Enter description for event No. ", $i + 1, ": ";
        $event_descriptions[$i][$j] = trim(fgets(STDIN));
    }
}

echo "Enter a country to search: ";
$needle = strtolower(trim(fgets(STDIN)));

$left = 0;
$right = EVENTS - 1;
```

```
$found = false;
while ($left <= $right && $found == false) {
    $middle = intval(($left + $right) / 2);

    if (strtolower($country_names[$middle]) > $needle) {
        $right = $middle - 1;
    }
    elseif (strtolower($country_names[$middle]) < $needle) {
        $left = $middle + 1;
    }
    else {
        $found = true;
        $index_position = $middle;
    }
}

if ($found == false) {
    echo "No country found!";
}
else {
    for ($i = 0; $i <= EVENTS - 1; $i++) {
        echo $event_descriptions[$i][$index_position], "\n";
    }
}

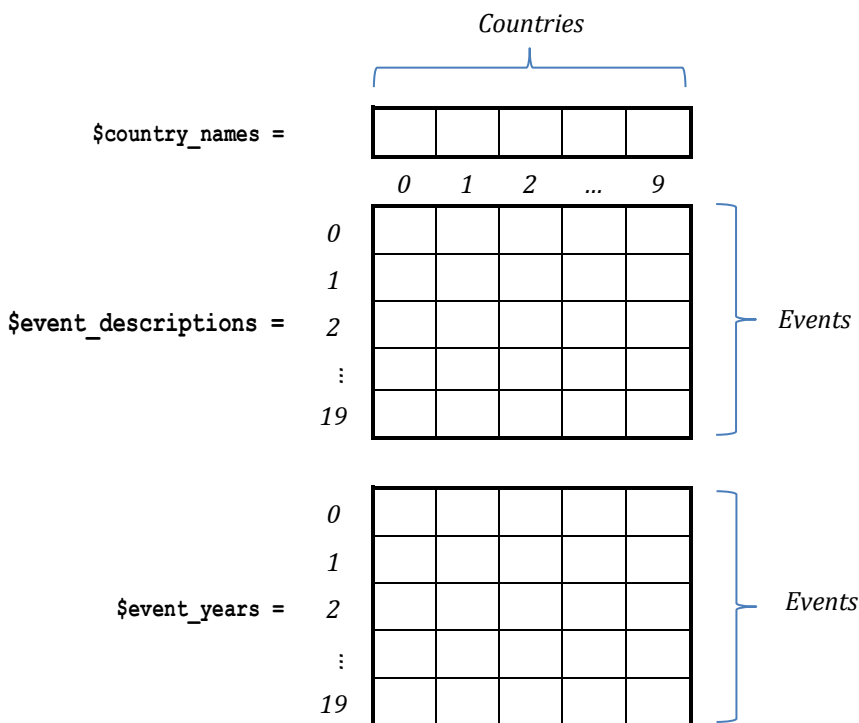
?>
```

Exercise 37.5-10 *Searching in Each Column of a Two-Dimensional Array*

Write a PHP script that prompts the user to enter the names of 10 countries and 20 important historical events for each country, including a brief description and the corresponding year of each event. The PHP script should then prompt the user to enter a year, and it will search and display all events that happened that year for each country. Use the binary search algorithm. Assume that for each country there is only one event in each year and that the user enters the events ordered by year in ascending order.

Solution

In this exercise, the following three arrays are required.



In order to write the code fragment that searches in each column of the array `$event_years`, let's use the "from inner to outer" method. The next binary search algorithm searches in the first column (column index 0) for a given year. Assume that variable `$j` contains the value 0. Since the search is performed vertically, and in order to increase script's readability, variables `$left` and `$right` have been replaced by variables `$top` and `$bottom` correspondingly.

```
$top = 0;
$bottom = EVENTS - 1;
$found = false;
while ($top <= $bottom && $found == false) {
    $middle = intval(($top + $bottom) / 2);

    if ($event_years[$middle][$j] > $needle) {
        $bottom = $middle - 1;
    }
    elseif ($event_years[$middle][$j] < $needle) {
        $top = $middle + 1;
    }
    else {
```

```

    $found = true;
    $row_index = $middle;
  }
}

if ($found == false) {
  echo "No event found for country ", $country_names[$j];
}
else {
  echo "Country: ", $country_names[$j] , "\n";
  echo "Year: ", $event_years[$row_index][$j], "\n";
  echo "Event: ", $event_descr[$row_index][$j], "\n\n";
}

```

Now, nesting this code fragment in a counted loop structure that iterates for all columns results in the following.

```

for ($j = 0; $j <= COUNTRIES - 1; $j++) {
  $stop = 0;
  $bottom = EVENTS - 1;
  $found = false;
  while ($stop <= $bottom && $found == false) {
    $middle = intval(($stop + $bottom) / 2);

    if ($event_years[$middle][$j] > $needle) {
      $bottom = $middle - 1;
    }
    elseif ($event_years[$middle][$j] < $needle) {
      $stop = $middle + 1;
    }
    else {
      $found = true;
      $row_index = $middle;
    }
  }

  if ($found == false) {
    echo "No event found for country ", $country_names[$j];
  }
  else {
    echo "Country: ", $country_names[$j] , "\n";
    echo "Year: ", $event_years[$row_index][$j], "\n";
  }
}

```

```
    echo "Event: ", $event_descr[$row_index][$j], "\n\n";
}
}
```

The final PHP script is as follows.

file_37.5-10.php

```
<?php
define("EVENTS", 20);
define("COUNTRIES", 10);

$country_names = array();
$event_descriptions = array(array());
$event_years = array(array());
for ($j = 0; $j <= COUNTRIES - 1; $j++) {
    echo "Enter Country No. ", $j + 1, ": ";
    $country_names[$j] = trim(fgets(STDIN));
    for ($i = 0; $i <= EVENTS - 1; $i++) {
        echo "Enter description for event No. ", $i + 1, ": ";
        $event_descriptions[$i][$j] = trim(fgets(STDIN));
        echo "Enter year for event No. ", $i + 1, ": ";
        $event_years[$i][$j] = trim(fgets(STDIN));
    }
}

echo "Enter a year to search: ";
$needle = trim(fgets(STDIN));

for ($j = 0; $j <= COUNTRIES - 1; $j++) {
    $top = 0;
    $bottom = EVENTS - 1;
    $found = false;
    while ($top <= $bottom && $found == false) {
        $middle = intval(($top + $bottom) / 2);

        if ($event_years[$middle][$j] > $needle) {
            $bottom = $middle - 1;
        }
        elseif ($event_years[$middle][$j] < $needle) {
            $top = $middle + 1;
        }
    }
}
```



```

    else {
        $found = true;
        $row_index = $middle;
    }
}

if ($found == false) {
    echo "No event found for country ", $country_names[$j];
}
else {
    echo "Country: ", $country_names[$j] , "\n";
    echo "Year: ", $event_years[$row_index][$j], "\n";
    echo "Event: ", $event_descriptions[$row_index][$j], "\n\n";
}
}
?>

```

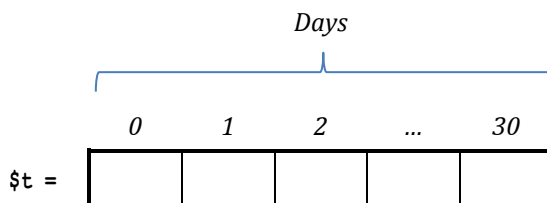
37.6 Exercises of a General Nature with Arrays

Exercise 37.6-1 *On Which Days was There a Possibility of Snow?*

Write a PHP script that lets the user enter the temperatures (in degrees Fahrenheit) recorded at 12:00 p.m. each day for the 31 days of January. The PHP script should then display the numbers of those days (1, 2, ..., 31) on which there was a possibility of snow, that is, those on which temperatures were below 36 degrees Fahrenheit (about 2 degrees Celsius).

Solution

The one-dimensional array for this exercise is as follows.



and the PHP script is as follows.

file_37.6-1.php

```

<?php
define("DAYS", 31);

```

```
$t = array();
for ($i = 0; $i <= DAYS - 1; $i++) {
    $t[$i] = trim(fgets(STDIN));
}

for ($i = 0; $i <= DAYS - 1; $i++) {
    if ($t[$i] < 36) {
        echo $i + 1, "\t";
    }
}

?>
```

Exercise 37.6-2 Was There Any Possibility of Snow?

Write a PHP script that lets the user enter the temperatures (in degrees Fahrenheit) recorded at 12:00 p.m. each day for the 31 days of January. The PHP script should then display a message indicating if there was a possibility of snow, that is, if there were any temperatures below 36 degrees Fahrenheit (about 2 degrees Celsius).

Solution

The code fragment that follows is **incorrect**. You **cannot** do what you did in the previous exercise.

```
for ($i = 0; $i <= DAYS - 1; $i++) {
    if ($t[$i] < 36) {
        echo "There was a possibility of snow in January!";
    }
}
```

If January had more than one day with a temperature below 36 degrees Fahrenheit, the same message would be displayed multiple times—and obviously you do not want this! You actually want to display a message once, regardless of whether January had one, two, or even more days below 36 degrees Fahrenheit.

There are two approaches, actually. Let's study them both.

First Approach – Counting all temperatures below 36 degrees Fahrenheit

In this approach, you can use a variable in the script to count all the days on which the temperature was below 36 degrees Fahrenheit. After all of the days have been examined, the script can check the value of this variable. If the value is not zero, it means that there was at least one day where there was a possibility of snow.

file_37.6-2a.php

```
<?php
define("DAYS", 31);
```

```
$t = array();
for ($i = 0; $i <= DAYS - 1; $i++) {
    $t[$i] = trim(fgets(STDIN));
}

$count = 0;
for ($i = 0; $i <= DAYS - 1; $i++) {
    if ($t[$i] < 36) {
        $count++;
    }
}

if ($count != 0) {
    echo "There was a possibility of snow in January!";
}

?>
```

Second Approach – Using a flag

In this approach, instead of counting all those days that had a temperature below 36 degrees Fahrenheit, you can use a Boolean variable (a flag). The solution is presented next.

file_37.6-2b.php

```
<?php
define("DAYS", 31);

$t = array();
for ($i = 0; $i <= DAYS - 1; $i++) {
    $t[$i] = trim(fgets(STDIN));
}

$found = false;
for ($i = 0; $i <= DAYS - 1; $i++) {
    if ($t[$i] < 36) {
        $found = true;
    }
}

if ($found == true) {
    echo "There was a possibility of snow in January!";
}
```

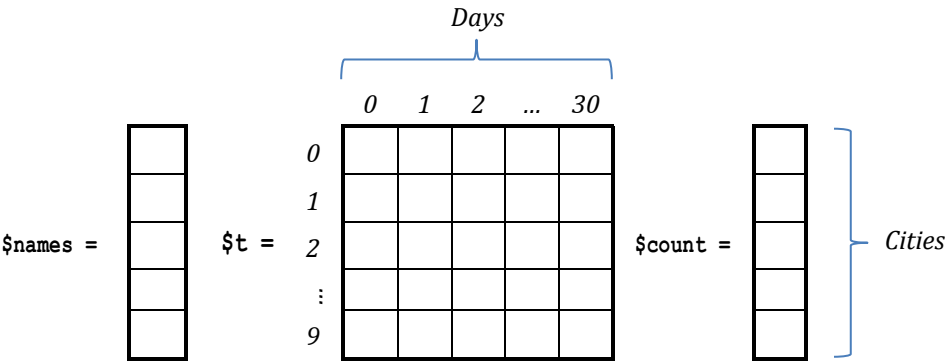
?>

Exercise 37.6-3 *In Which Cities was There a Possibility of Snow?*

Write a PHP script that prompts the user to enter the names of ten cities and their temperatures (in degrees Fahrenheit) recorded at 12:00 p.m. each day for the 31 days of January. The PHP script should display the names of the cities in which there was a possibility of snow, that is, those in which temperatures were below 36 degrees Fahrenheit (about 2 degrees Celsius).

Solution

This exercise needs two arrays: one for entering the names of the cities, and one for entering their corresponding temperatures. The third auxiliary array (`$count`), presented next, can be created by the script to count the total number of days on which each city had temperatures lower than 36 degrees Fahrenheit. Of course, you may wonder now why you need this extra array `$count`. Just be patient! You will soon find out why!



As in the previous exercise, you need to display each city name once, regardless of whether it may had one, two, or even more days below 36 degrees Fahrenheit. To solve this exercise, there are actually two approaches. The first approach uses the previously mentioned auxiliary array `$count`, whereas the second one uses just one extra Boolean variable (a flag). Obviously the second one is more efficient. But let's study them both.

First Approach – Using an auxiliary array

You were taught in Chapter 35 how to process each row individually. The nested loop control structure that can create the auxiliary array `$count` is as follows.

```
$count = array();  
for ($i = 0; $i <= CITIES - 1; $i++) {  
    $count[$i] = 0;  
    for ($j = 0; $j <= DAYS - 1; $j++) {  
        if ($a[$i][$j] < 36) {
```

```

        $count[$i]++;
    }
}
}

```

After array `$count` is created you can iterate through it, and when a position contains a value other than zero, it means that the corresponding city had at least one day below 36 degrees Fahrenheit; thus the script must display the name of that city. The final PHP script is presented next

file_37.6-3a.php

```

<?php
define("CITIES", 10);
define("DAYS", 31);

$names = array();
$t = array(array());
for ($i = 0; $i <= CITIES - 1; $i++) {
    echo "Enter a name for city No: ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= DAYS - 1; $j++) {
        echo "Enter a temperature for day No: ", $j + 1, ": ";
        $t[$i][$j] = trim(fgets(STDIN));
    }
}

$count = array();
for ($i = 0; $i <= CITIES - 1; $i++) {
    $count[$i] = 0;
    for ($j = 0; $j <= DAYS - 1; $j++) {
        if ($t[$i][$j] < 36) {
            $count[$i]++;
        }
    }
}

echo "Cities in which there was a possibility of snow in January: ";
for ($i = 0; $i <= CITIES - 1; $i++) {
    if ($count[$i] != 0) {
        echo $names[$i], "\n";
    }
}
}

```

```
?>
```

Second Approach – Using a flag

This approach does not use an auxiliary array. It processes array `$t` and directly displays any city name that had a temperature below 36 degrees Fahrenheit. But how can this be done without displaying a city name twice, or even more than twice? This is where you need a flag, that is, an extra Boolean variable.

To better understand this approach, let's use the “from inner to outer” method. The following code fragment checks if the first row of array `$t` (row index 0) contains at least one temperature below 36 degrees Fahrenheit; if so, it displays the corresponding city name that exists at position 0 of the array `$names`. Assume variable `$i` contains the value 0.

```
$found = false;
for ($j = 0; $j <= DAYS - 1; $j++) {
    if ($t[$i][$j] < 36) {
        $found = true;
    }
}

if ($found == true) {
    echo $names[$i];
}
```

Now that everything has been clarified, in order to process the whole array `$t`, you can just nest this code fragment in a counted loop structure that iterates for all cities, as follows.

```
for ($i = 0; $i <= CITIES - 1; $i++) {
    $found = false;
    for ($j = 0; $j <= DAYS - 1; $j++) {
        if ($t[$i][$j] < 36) {
            $found = true;
        }
    }

    if ($found == true) {
        echo $names[$i];
    }
}
```

The final PHP script is as follows.

`file_37.6-3b.php`

```
<?php
```

```
define("CITIES", 10);
```

```
define("DAYS", 31);

$names = array();
$t = array(array());
for ($i = 0; $i <= CITIES - 1; $i++) {
    echo "Enter a name for city No: ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= DAYS - 1; $j++) {
        echo "Enter a temperature for day No: ", $j + 1, ": ";
        $t[$i][$j] = trim(fgets(STDIN));
    }
}

echo "Cities in which there was a possibility of snow in January: ";
for ($i = 0; $i <= CITIES - 1; $i++) {
    $found = false;
    for ($j = 0; $j <= DAYS - 1; $j++) {
        if ($t[$i][$j] < 36) {
            $found = true;
        }
    }

    if ($found == true) {
        echo $names[$i];
    }
}

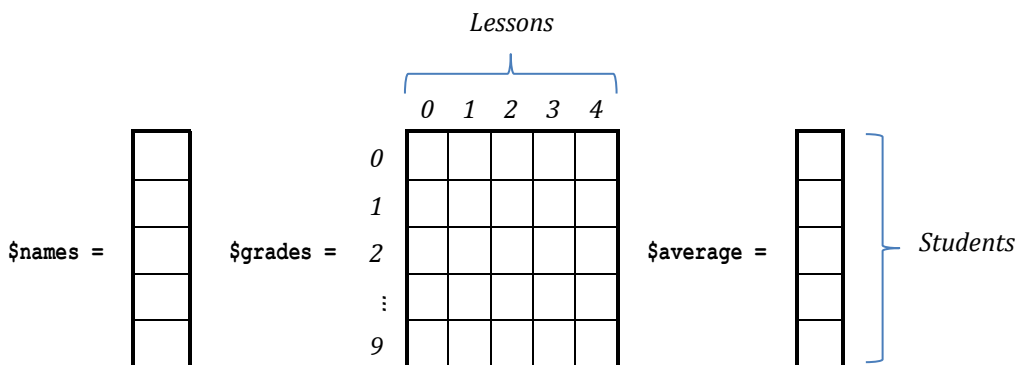
?>
```

Exercise 37.6-4 *Display from Highest to Lowest Grades by Student, and in Alphabetical Order*

There are 10 students and each one of them has received his or her grades for five lessons. Write a PHP script that lets a teacher enter the name of each student and his or her grades for all lessons. The script should then calculate each student's average grade, and display the names of the students sorted by their average grade in descending order. Moreover, if two or more students have the same average grade, their names should be displayed in alphabetical order.

Solution

In this exercise, you need the following three arrays. The values for the arrays `$names` and `$grades` will be entered by the user, whereas the auxiliary array `$average` will be created by the PHP script.



You already know how to do everything in this exercise. You know how to create the auxiliary array `$average`, you know how to sort the array `$average` while preserving the relationship with the elements of the array `$names`, and you know how to handle the case in which, if two average grades are equal, the corresponding student names should be sorted alphabetically. The final PHP script is as follows.

file_37.6-4.php

```
<?php
define("STUDENTS", 10);
define("LESSONS", 5);

//Read array $names and $grades
$names = array();
$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "Enter name for student No. ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        echo "Enter grade for lesson No. ", $j + 1, ": ";
        $grades[$i][$j] = trim(fgets(STDIN));
    }
}

//Create array $average
$average = array();
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    $average[$i] = 0;
```



```

    for ($j = 0; $j <= LESSONS - 1; $j++) {
        $saverage[$i] += $grades[$i][$j];
    }
    $saverage[$i] /= LESSONS;
}

//Sort arrays $names and $saverage
for ($m = 1; $m <= STUDENTS - 1; $m++) {
    for ($n = STUDENTS - 1; $n >= $m; $n--) {
        if ($saverage[$n] > $saverage[$n - 1]) {
            $temp = $saverage[$n];
            $saverage[$n] = $saverage[$n - 1];
            $saverage[$n - 1] = $temp;

            $temp = $names[$n];
            $names[$n] = $names[$n - 1];
            $names[$n - 1] = $temp;
        }
        elseif ($saverage[$n] == $saverage[$n - 1]) {
            if ($names[$n] < $names[$n - 1]) {
                $temp = $names[$n];
                $names[$n] = $names[$n - 1];
                $names[$n - 1] = $temp;
            }
        }
    }
}

//Display arrays $names and $saverage
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo $names[$i], "\t", $saverage[$i], "\n";
}
?>

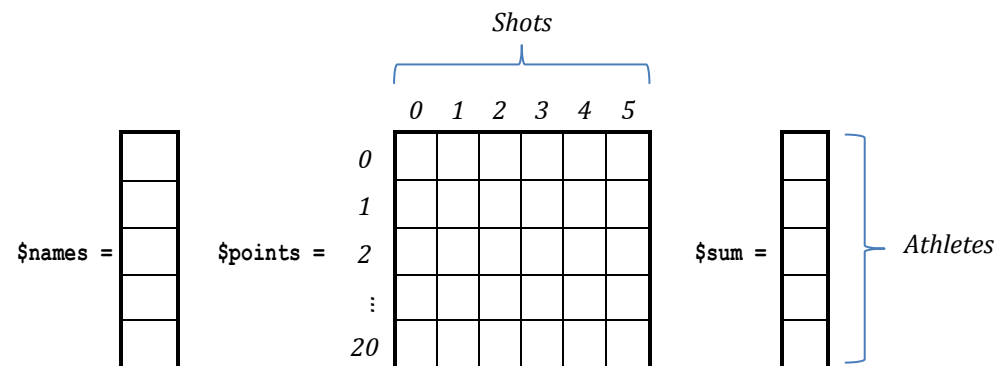
```

Exercise 37.6-5 Archery at the Summer Olympics

In archery at the Summer Olympics, 20 athletes each shoot six arrows. Write a PHP script that prompts the user to enter the name of each athlete, and the points awarded for each shot. The script should then display the names of the three athletes that won the gold, silver, and bronze medals depending on which athlete obtained the highest sum of points. Assume that no two athletes have an equal sum of points.

Solution

In this exercise, you need the following three arrays. The values for the arrays `$names` and `$points` will be entered by the user, whereas the auxiliary array `$sum` will be created by the PHP script



After the auxiliary array `$sum` is created, a sorting algorithm can sort the array `$sum` in descending order (while preserving the relationship with the elements of the array `$names`). The PHP script can then display the names of the three athletes at index positions 0, 1, and 2 (since these are the athletes that should win the gold, the silver, and the bronze medals respectively). The solution is presented next.

file_37.6-5.php

```
<?php
define("ATHLETES", 20);
define("SHOTS", 6);

//Read array $names and $points
$names = array();
$points = array(array());
for ($i = 0; $i <= ATHLETES - 1; $i++) {
    echo "Enter name for athlete No. ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));
    for ($j = 0; $j <= SHOTS - 1; $j++) {
        echo "Enter points for shot No. ", $j + 1, ": ";
        $points[$i][$j] = trim(fgets(STDIN));
    }
}

//Create array $sum
$sum = array();
```

```

for ($i = 0; $i <= ATHLETES - 1; $i++) {
    $sum[$i] = 0;
    for ($j = 0; $j <= SHOTS - 1; $j++) {
        $sum[$i] += $points[$i][$j];
    }
}

//Sort arrays $names and $sum
for ($m = 1; $m <= ATHLETES - 1; $m++) {
    for ($n = ATHLETES - 1; $n >= $m; $n--) {
        if ($sum[$n] > $sum[$n - 1]) {
            $temp = $sum[$n];
            $sum[$n] = $sum[$n - 1];
            $sum[$n - 1] = $temp;

            $temp = $names[$n];
            $names[$n] = $names[$n - 1];
            $names[$n - 1] = $temp;
        }
    }
}

//Display gold, silver and bronze metal
for ($i = 0; $i <= 2; $i++) {
    echo $names[$i], "\t", $sum[$i], "\n";
}
?>

```

37.7 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. The main idea of the bubble sort algorithm (when sorting an array in ascending order) is to repeatedly move the smallest elements of the array to the lowest index positions.
2. In an array sorted in ascending order, the first element is the greatest of all.
3. When using the bubble sort algorithm, the total number of swaps depends on the given array.
4. When using the modified bubble sort algorithm, the case in which the algorithm performs the greatest number of swaps is when you want to sort in descending order an array that is already sorted in ascending order.

5. In the bubble sort algorithm, when the decision control structure tests the Boolean expression $A[n] > A[n - 1]$, it means that the elements of array A are being sorted in descending order.
6. Sorting algorithms sort letters in the same way that they sort numbers.
7. If you want to sort an array A but preserve the relationship with the elements of an array B, you must rearrange the elements of array B as well.
8. The bubble sort algorithm sometimes performs better than the modified bubble sort algorithm.
9. According to the bubble sort algorithm, in each pass (except the last one) only one element is placed in proper position.
10. The bubble sort algorithm can be implemented only by using counted loop structures.
11. The quick sort algorithm cannot be used to sort each column of a two-dimensional array.
12. The insertion sort algorithm can sort in either descending or ascending order.
13. One of the fastest sorting algorithms is the modified bubble sort algorithm.
14. The bubble sort algorithm, for a one-dimensional array of N elements, performs $\frac{N-1}{N}$ compares.
15. The bubble sort algorithm, for a one-dimensional array of N elements, performs $\frac{N(N-1)}{2}$ passes.
16. When using the modified bubble sort algorithm, if a complete pass is performed and no swaps have been done, then the algorithm knows the array is sorted and there is no need for further passes.
17. When using the selection sort algorithm, if you wish to sort an array in descending instead of ascending order, you need to search for maximum instead of minimum values.
18. The selection sort algorithm performs very well on computer systems in which the limited main memory comes into play.
19. The selection sort algorithm is suitable for large scale data operations.
20. The selection sort algorithm is a very complex algorithm.
21. The insertion sort algorithm generally performs better than the selection and the bubble sort algorithm.
22. The insertion sort algorithm can sometimes prove even faster than the quicksort algorithm.
23. The quicksort algorithm is considered one of the best and fastest sorting algorithms.
24. A sorted array contains only elements that are different from each other.
25. A search algorithm is an algorithm that searches for an item with specific properties within a set of data.

26. The sequential search algorithm can be used only on arrays that contain arithmetic values.
27. One of the most commonly used search algorithms is the quick search algorithm.
28. One searching algorithm is called the heap algorithm.
29. A linear (or sequential) search algorithm can work as follows: it can check if the last element of the array is equal to a given value, then it can check the last but one element, and so on, until the beginning of the array or until the given value is found.
30. The linear search algorithm can, in certain situations, find an element faster than the binary search algorithm.
31. The linear search algorithm can be used in large scale data operations.
32. The linear search algorithm cannot be used in sorted arrays.
33. The binary search algorithm can be used in large scale data operations.
34. If an array contains a value multiple times, the binary search algorithm can find only the first occurrence of a given value.
35. When using search algorithms, if an array contains unique values and the element that you are looking for is found, there is no need to check any further.
36. The main disadvantage of the binary search algorithm is that data needs to be sorted.
37. The binary search algorithm can be used only in arrays that contain arithmetic values.
38. If the element that you are looking for is in the last position of an array, the linear search algorithm will examine all the elements of the array.
39. The linear search algorithm can be used on two-dimensional arrays.
40. When using the binary search algorithm, if the element that you are looking for is at the first position of an array that contains at least three elements, the algorithm will find it in just one iteration.

37.8 Review Exercises

Complete the following exercises.

1. Design a flowchart and write the corresponding PHP script that lets the user enter 50 positive numerical values into an array. The algorithm, and consequently the PHP script, should then create a new array of 47 elements. In this new array, each position should contain the average value of four elements: the values that exist in the current and the next three positions of the given array.
2. Write a PHP script that lets the user enter numerical values into arrays `$a`, `$b`, and `$c`, of 15 elements each. The script should then create a new array `$new` of 15 elements. In this new array, each position should contain the lowest value of arrays `$a`, `$b`, and `$c`, for the corresponding position.

Next, design the corresponding flowchart fragment for only that part of your script that creates the array `$new`.

3. Write a PHP script that lets the user enter numerical values into arrays `$a`, `$b`, and `$c`, of 10, 5, and 15 elements respectively. The script should then create a new array `$new` of 30 elements. In this new array, the first 15 positions should contain the elements of array `$c`, the next five positions should contain the elements of array `$b`, and the last 10 positions should contains the elements of array `$a`.

Next, design the corresponding flowchart fragment for only that part of your script that creates the array `$new`.

4. Write a PHP script that lets the user enter numerical values into arrays `$a`, `$b`, and `$c`, of 5×10 , 5×15 , and 5×20 elements respectively. The script should then create a new array `$new` of 5×45 elements. In this new array, the first 10 columns should contain the elements of array `$a`, the next 15 columns should contain the elements of array `$b`, and the last 20 rows should contain the elements of array `$c`.
5. Write a PHP script that lets the user enter 50 numerical values into an array and then creates two new arrays, `$reals` and `$integers`. The array `$reals` should contain the real values given, whereas the array `$integers` should contain the integer values. The value 0 (if any) should not be added to any of the final arrays, either `$reals` or `$integers`.

Next, design the corresponding flowchart fragment for only that part of your script that creates the arrays `$reals` and `$integers`.

6. Design a flowchart and write the corresponding PHP script that lets the user enter 50 three-digit integers into an array. The algorithm, and consequently the PHP script, then creates a new array containing only the integers in which the first digit is less than the second digit and the second digit is less than the third digit. For example, the values 357, 456, and 159 are such integers.
7. A public opinion polling company asks 200 citizens to each score 10 consumer products. Write a PHP script that prompts the user to enter the name of each product and the score each citizen gave (A, B, C, or D). The script should then calculate and display the following:
 - a. for each product, the name of the product and the number of citizens that gave it an "A"
 - b. for each citizen, the number of "B" responses he or she gave
 - c. which product or products are considered the best

Moreover, the script should validate data input and display an error message when the user enters any score with a value other than A, B, C, or D.

8. Write a PHP script that prompts the user to enter the names of 20 U.S. cities and the names of 20 Canadian cities and then, for each U.S. city, the distance (in miles) from each Canadian city. Finally, the script should display, for each U.S. city, its closest Canadian city.

9. Design a flowchart and write the corresponding PHP script that lets the user enter the names and the heights of 30 mountains, as well as the country in which each one belongs. The algorithm, and consequently the PHP script, should then display all available information about the highest and the lowest mountain.
10. Design the flowchart fragment of an algorithm that, for a given array A of $N \times M$ elements, finds and displays the maximum value as well as the row and the column in which this value was found.
11. Twenty-six teams participate in a football tournament. Each team plays 15 games, one game each week. Write a PHP script that lets the user enter the name of each team and the letter “W” for win, “L” for loss, and “T” for tie (draw) for each game. If a win receives 3 points and a tie 1 point, the PHP script should find and display the name of the team that wins the championship based on which team obtained the greatest sum of points. Assume that no two teams have an equal sum of points.
12. On Earth, a free-falling object has an acceleration of 9.81 m/s^2 downward. This value is denoted by g . A student wants to calculate that value using an experiment. She allows 10 different objects to fall downward from a known height, and measures the time they need to reach the floor. She does this 20 times for each object. She needs a PHP script that allows her to enter the height (from which objects are left to fall), as well as the measured times that they take to reach the floor. The script should then calculate g and store all calculated values in a 10×20 array. However, her chronometer is not so accurate, so she needs the script to find and display the minimum and the maximum calculated values of g for each object, as well as the overall minimum and maximum calculated values of g of all objects.

The required formula is

$$S = u_0 + \frac{1}{2}gt^2$$

where

- S is the distance that the free-falling objects traveled, in meters (m)
 - u_0 is the initial velocity (speed) of the free-falling objects in meters per second (m/sec). However, since the free-falling objects start from rest, the value of u_0 should be zero.
 - t is the time that it took the free-falling object to reach the floor, in seconds (sec)
 - g is the acceleration, in meters per second² (m/sec²)
13. Ten measuring stations, one in each city, record the daily CO_2 levels for a period of a year. Write a PHP script that lets the user enter the name of each city and the CO_2 levels recorded at 12:00 p.m. each day. The PHP script then displays the name of the city that has the clearest atmosphere (on average).
 14. Design the flowchart fragment of an algorithm that, for a given array A of $N \times M$ elements, finds and displays the minimum and the maximum values of each row.

15. Twenty teams participate in a football tournament, and each team plays 10 games, one game each week. Write a PHP script that prompts the user to enter the name of each team and the letter “W” for win, “L” for loss, and “T” for tie (draw) for each game. If a win receives 3 points and a tie 1 point, the PHP script should find and display the names of the teams that win the gold, the silver, and the bronze medals based on which team obtained the greatest sum of points. Use the modified bubble sort algorithm. Assume that no two teams have an equal sum of points.

Moreover, the script should validate data input and display an error message when the user enters any letter other than W, L, or T.

16. Write a PHP script that prompts the user to enter the names and the heights of 50 people. The script should then display this information, sorted by height, in descending order. Moreover, if two or more people have the same height, their names should be displayed in alphabetical order. Use the bubble sort algorithm, adapted accordingly.
17. In a song contest there are 10 judges, each of whom scores 12 artists for their performance. However, according to the rules of this contest, the total score is calculated after excluding the maximum and the minimum score. Write a PHP script that prompts the user to enter the names of the artists and the score they get from each judge. The script should then display
- for each artist, his or her name and total score, after excluding the maximum and the minimum scores.
 - the final classification, starting with the artist that has the greatest score. However, if two or more artists have the same score, their names should be displayed in alphabetical order. Use the bubble sort algorithm, adapted accordingly.
18. Design the flowchart fragment of an algorithm that, for a given array A of 5×10 elements, sorts each column in ascending order using the selection sort algorithm.
19. In a Sudoku contest, 10 people each try to solve as quickly as possible eight different Sudoku puzzles. Write a PHP script that lets the user enter the name of each contestant and their time to complete each puzzle. The script should then display
- for each contestant, his or her name and his or her three best times.
 - the names of the three contestants that win the gold, the silver, and the bronze medals based on which contestant obtained the lowest average time. Assume that no two contestants have an equal average time.

Use the selection sort algorithm when necessary.

20. Design the flowchart fragment of an algorithm that, for a given array A of 20×8 elements, sorts each row in descending order using the insertion sort algorithm.

21. Five measuring stations, one in each city, record the daily CO₂ levels on an hourly basis for a period of two days. Write a PHP script that lets the user enter the name of each city and the CO₂ levels recorded every hour (00:00 to 23:00). The PHP script then displays
 - a. for each city, its name and its average CO₂ level (per hour)
 - b. for each hour, the average CO₂ level (per city)
 - c. the hour in which the atmosphere was most polluted (on average)
 - d. the hour and the city in which the highest level of CO₂ was recorded
 - e. the three cities with the dirtiest atmosphere (on average), using the insertion sort algorithm
22. Design the flowchart fragment of the linear search algorithm that searches array `$a` of `N` elements for the value `$needle` and displays the position(s) at which `$needle` is found. If `$needle` is not found, the message “Not found” should be displayed.
23. Design the flowchart fragment of the binary search algorithm that searches array `$a` of `N` elements for the value `$needle` and displays the position at which `$needle` is found. If `$needle` is not found, the message “Not found” should be displayed.
24. Ten teams participate in a football tournament, and each team plays 16 games, one game each week. Write a PHP script that prompts the user to enter the name of each team, the number of goals the team scored, and the number of goals the team let in for each match. A win receives 3 points and a tie receives 1 point. The PHP script should prompt the user for a team name and then it finds and displays the total number of points for this team. If the given team name is not found, the message “This team does not exist” should be displayed.

Assume that no two teams have the same name. Moreover, the script should validate data input and display an error message when the user enters any non-numeric or negative number of goals.

25. In a high school, there are two classes, with 20 and 25 students respectively. Write a PHP script that prompts the user to enter the names of the students in two separate arrays. The script then displays the names of each class independently in ascending order. Afterwards, the script prompts the user to enter a name and it searches for that given name in both arrays. If the student's name is found, the script should display the message “Student found in class No...”; otherwise the message “Student not found in either class” should be displayed. Assume that both arrays contain unique names.
Hint: Since the arrays are sorted and the names are unique, you can use the binary search algorithm.
26. Suppose there are two arrays, `$usernames` and `$passwords`, that contain the login information of 100 employees of a company. Design a flowchart fragment and write the corresponding code fragment that prompts the user to enter a username and a password and then displays the message “Login

OK!” when the combination of username and password is valid; the message “Login Failed!” should be displayed otherwise. Assume that usernames are unique but passwords are not. Moreover, both usernames and passwords are not case sensitive.

27. Suppose there are two arrays, `$names` and `$SSNs`, that contain the names and the SSNs (Social Security Numbers) of 1000 U.S. citizens. Write a code fragment that prompts the user to enter a value (it can be either a name or an SSN) and then searches for and displays the names of all the people that have this name or this SSN. If the given value is not found, the message “This value does not exist” should be displayed.
28. There are 12 students and each one of them has received his or her grades for six lessons. Write a PHP script that lets the user enter the grades for all lessons and then displays a message indicating whether or not there is at least one student that has an average value below 70.

Moreover, the script should validate data input and display individual error messages when the user enters any non-numeric character, negative value, or a value greater than 100.

Review Questions in “Arrays”

Answer the following questions.

1. What limitation do variables have that arrays don't?
2. What is an array?
3. What is each item of an array called?
4. In an array of 100 elements, what is the index of the last element?
5. What does “iterating through rows” mean?
6. What does “iterating through columns” mean?
7. What is the significance to programmers of variables i and j ?
8. What is a square array?
9. What is the main diagonal of a square array?
10. What is the antidiagonal of a square array?
11. Write the code fragment in general form that validates data input without displaying any error messages.
12. Write the code fragment in general form that validates data input and displays an error message, regardless of the type of error.
13. Write the code fragment in general form that validates data input and displays an individual error message for each error.
14. What is a sorting algorithm?
15. Name five sorting algorithms.
16. Which sorting algorithm is considered the most inefficient?
17. Can a sorting algorithm be used to find the minimum or the maximum value of an array?
18. Why is a sorting algorithm not the best option to find the minimum or the maximum value of an array?
19. Write the code fragment that sorts array a of N elements in ascending order, using the bubble sort algorithm.
20. How many compares can the bubble sort algorithm perform?
21. When does the bubble sort algorithm perform the maximum number of swaps?
22. Using the bubble sort algorithm, write the code fragment that sorts array a but preserves the relationship with the elements of array b of N elements in ascending order.
23. Using the modified bubble sort algorithm, write the code fragment that sorts array a of N elements in ascending order.
24. Using the selection sort algorithm, write the code fragment that sorts array a of N elements in ascending order.

25. Using the insertion sort algorithm, write the code fragment that sorts array `$a` of `N` elements in ascending order.
26. What is a searching algorithm?
27. Name the two most commonly used search algorithms.
28. What are the advantages and disadvantages of the linear search algorithm?
29. Using the linear search algorithm, write the code fragment that searches array `$a` for value `$needle`.
30. What are the advantages and disadvantages of the binary search algorithm?
31. Using the binary search algorithm, write the code fragment that searches array `$a` for value `$needle`. Assume array `$a` is sorted in ascending order.

Section 7

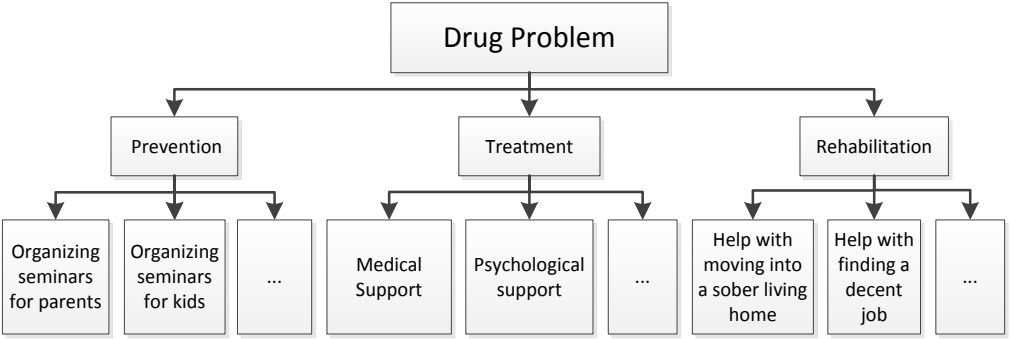
Subprograms

Chapter 38

Introduction to Subprograms

38.1 What is Procedural Programming?

Suppose you were assigned a project to solve the drug abuse problem in your area. One possible approach, which could prove very difficult or even impossible, would be to try to solve this problem by yourself! A better approach would be to subdivide the large problem into smaller subproblems such as prevention, treatment, and rehabilitation. Then, with the help of specialists from a variety of fields, you would build a team to help solve the drug problem. The following diagram shows how each subproblem could be further subdivided into even smaller subproblems.



As the supervisor of this project, you could rent a building and establish within it three departments: the prevention department, with all of its subdepartments; the treatment department, with all of its subdepartments; and the rehabilitation department with all of its subdepartments. Finally, you would hire staff and employ them to do the job for you!

Procedural programming does exactly the same thing. It subdivides an initial problem into smaller subproblems, and each subproblem is further subdivided into smaller subproblems. Finally, for each subproblem a small subprogram would be written, and the main program (as does the supervisor), employs (calls) them to do job!

Procedural programming has several advantages:

- It enables programmers to reuse the same code whenever it is necessary without having to copy it.
- It is relatively easy to implement.
- It helps programmers follow the flow of execution more easily.

Notice: A very large program can prove very difficult to debug and maintain when it is all in one piece. For this reason, it is often easier to subdivide it into smaller subprograms, each of which performs a clearly defined process.

Notice: Writing large programs without subdividing them into smaller subprograms results in a code referred to as “spaghetti code”!

38.2 What is Modular Programming?

In modular programming, subprograms of common functionality are grouped together into separate modules, and each module can have its own set of data. Therefore, a program can consist of more than one part, and each of those smaller parts (modules) can contain one or more subprograms.

If you were to use modular programming in the previous drug problem example, then you could have one building to host the prevention department and all of its subdepartments, a second building to host the treatment department and all of its subdepartments, and a third building to host the rehabilitation department and all of its subdepartments. As shown in the next diagram, the three buildings could be thought as three different modules.



38.3 What Exactly is a Subprogram?

In computer science, a subprogram is a block of statements packaged as a unit that performs a specific task. A subprogram can be employed (called) in a script or program several times, whenever that specific task needs to be performed.

A built-in PHP function is an example of such a subprogram. Take the already known `abs()` function, for example. It consists of a block of statements that are packaged as a unit under the name “abs,” and they perform a specific task—they return the absolute value of a number.

Notice: If you are wondering what type of statements might exist inside unit `abs()`, here is a possible block of statements.

```
if ($number < 0)
    return $number * (-1);
else
    return $number;
```

Generally speaking, there are two types of subprograms: functions and procedures. The difference between a function and a procedure is that a function returns a result, whereas a procedure doesn't. However, in some computer languages, this distinction may not quite be apparent. There are languages in which a function can behave as a procedure and return no result (PHP is such a language), and there are languages in which a procedure can return one or even more than one result.

Depending on the computer language being used, the meaning of the terms “functions” and “procedures” may vary. For example, in Visual Basic you can find them as “functions” and “subprocedures,” whereas in FORTRAN you would find them as “functions” and “subroutines.” On the other hand, PHP supports only functions that can play both roles; they can behave either as a function or as a procedure depending on the way they are written.

38.4 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Procedural programming helps you write “spaghetti code.”
2. Procedural programming subdivides the initial problem into smaller subproblems.
3. An advantage of procedural programming is the ability to reuse the same code whenever it is necessary without having to copy it.
4. Procedural programming helps programmers follow the flow of execution more easily.
5. Modular programming improves a program’s execution speed.
6. In modular programming, subprograms of common functionality are grouped together into separate modules.
7. In modular programming, each module can have its own set of data.
8. Modular programming uses different structures than structured programming does.
9. A program can consist of more than one module.
10. A subprogram is a block of statements packaged as a unit that performs a specific task.
11. The difference between a function and a procedure is that a procedure returns a result, whereas a function does not.
12. There are two types of subprograms.
13. PHP supports only procedures.

Chapter 39

User-Defined Functions

39.1 Writing your Own Functions in PHP

In PHP, and by extension in the majority of computer languages, there are two kinds of functions. There are the built-in functions, such as `trim()`, `strpos()`, `sin()`, and there are the user-defined functions, those that you can actually write and use in your own scripts.

The general form of a PHP function is

```
function name([arg1, arg2, arg3, ...]) {  
    A statement or block of statements  
    return value;  
}
```

where

- *name* is the name of the function. It follows the same rules as those used for variable names
- *arg1, arg2, arg3, ...* is a list of arguments (variables) used to pass values from the caller to the function. There can be as many arguments as you want.
- *value* is the arithmetic, alphanumeric, or Boolean value that is returned to the caller. It can be a constant value, a variable, an expression or even an array.

Notice: Please note that arguments are optional; that is, a function may contain no arguments.

For example, the next function calculates the sum of two numbers and returns the result.

```
function get_sum($num1, $num2) {  
    $result = $num1 + $num2;  
    return $result;  
}
```

Of course, this can also be written as

```
function get_sum($num1, $num2) {  
    return $num1 + $num2;  
}
```

39.2 How Do You Call a Function?

Every call to a user-defined function is made in the same way as calling the built-in functions of PHP. You write the name of the function followed by a list of arguments (if necessary), either within a statement that assigns function's returned value to a variable or directly within an expression.

Let's see some examples. Suppose there is a function `cube_root()` that accepts an argument (a numeric value) and returns its 3rd root. To calculate the expression

$$y = \sqrt[3]{x} + \frac{1}{x}$$

you can either assign the cube root of x to a variable, as shown here

```
$x = trim(fgets(STDIN));
$cb = cube_root($x);
$y = $cb + 1 / $x;
echo $y;
```

or you can call the function directly in an expression,

```
$x = trim(fgets(STDIN));
$y = cube_root($x) + 1 / $x;
echo $y;
```

or you can even call the function directly in an `echo` statement.

```
$x = trim(fgets(STDIN));
echo cube_root($x) + 1 / $x;
```

Now let's see a more complete example. The next PHP script creates the function `get_message()` and then the main code calls it. The returned value is assigned to variable `$a`.

file_39.2a.php

```
<?php
function get_message() {
    $msg = "Hello Zeus";
    return $msg;
}

echo "Hi there!";
$a = get_message();
echo $a;
?>
```

This creates the function `get_message()`

This is the main code

Notice: Please note that a function does not execute immediately when a script starts running. The first statement that actually executes in this example is the statement `echo "Hi there!"`

You can also pass (send) values to a function, as long as at least one argument exists within the function's parenthesis. In the next example, the function `display()` is called three times but each time a different value is passed through the argument `$color`.

file_39.2b.php

```
<?php
function display($color) {
```

```
$msg = "There is " . $color . " in the rainbow\n";  
return $msg;  
}  
  
echo display("red");  
echo display("yellow");  
echo display("blue");  
?>
```

If you run this script, the following messages are displayed.

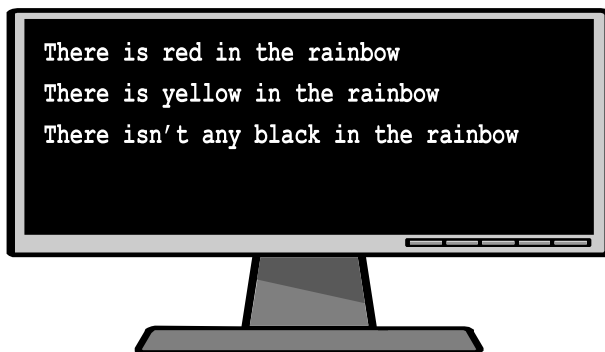


In the next example, two values must be passed to function `display()`.

file_39.2c.php

```
<?php  
function display($color, $exists) {  
    $neg = "";  
    if ($exists == false) {  
        $neg = "n't any";  
    }  
  
    return "There is" . $neg . " " . $color . " in the rainbow\n";  
}  
  
echo display("red", true);  
echo display("yellow", true);  
echo display("black", false);  
?>
```

If you run this script the following messages are displayed.



Notice: You can place your functions either above or below your main code. Most programmers, though, prefer to have them all on the top for better observation.

39.3 Formal and Actual Function Arguments

Each function contains an argument list called a “formal argument list.” As already stated, arguments in this list are optional; the formal argument list may contain no arguments, one argument, or more than one argument.

When the function is called, an argument list is passed to the function. This list is called an “actual argument list.”

In the next example, variables `$number1`, `$number2`, `$number3`, and `$number4` constitute the formal argument list whereas variables `$a`, `$b`, `$c`, and `$d` constitute the actual argument list.

file_39.3.php

```
<?php
```

```
function add($number1, $number2, $number3, $number4) {  
  
    $result = $number1 + $number2 + $number3 + $number4;  
    return $result;  
}
```

This is the formal argument list

```
$a = trim(fgets(STDIN));  
$b = trim(fgets(STDIN));  
$c = trim(fgets(STDIN));  
$d = trim(fgets(STDIN));
```

This is the actual argument list

```
$sum = add($a, $b, $c, $d);
```

```
echo $sum;
?>
```

Notice: Please note that there is a one-to-one match between the formal and the actual arguments. The value of argument `$a` is passed to argument `$number1`, the value of argument `$b` is passed to argument `$number2`, and so on.

39.4 How Does a Function Execute?

When the main code calls a function the following steps are performed:

- The execution of the statements of the main code is interrupted.
- The values of the variables or the result of the expressions that exist in the actual argument list are passed (assigned) to the corresponding arguments (variables) in the formal argument list, and the flow of execution goes to where the function is written.
- The statements of the function are executed.
- When the flow of execution reaches the end of the function, a value is returned from the function to the main code and the flow of execution continues from where it was before calling the function.

In the next PHP script, the function `maximum()` accepts two arguments (numeric values) and returns the greater of the two values.

file_39.4.php

```
<?php
function maximum($val1, $val2) {
    $m = $val1;
    if ($val2 > $m) {
        $m = $val2;
    }
    return $m;
}

$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$max = maximum($a, $b);
echo $max;
?>
```

When the PHP script starts running, the first statement executed is the statement `$a = trim(fgets(STDIN))` (this is considered the first statement of the script). Suppose the user enters the values 3 and 8. Below is a trace table that shows the exact flow of execution, how the values of the variables `$a` and `$b` are passed from the main code to the function, and how the function returns its result.

Step	Statements of Main Code	\$a	\$b	\$max
1	\$a = trim(fgets(STDIN))	3	?	?
2	\$b = trim(fgets(STDIN))	3	8	?
3	\$max = maximum(\$a, \$b)			

When a call to the function `maximum()` is made, the execution of the statements of the main code is interrupted and the values of the variables `$a` and `$b` are copied (assigned) to the corresponding arguments (variables) `$val1` and `$val2` and the flow of execution goes to where the function is written. Then the statements of the function are executed.

Step	Statements of Function <code>maximum()</code>	<code>\$val1</code>	<code>\$val2</code>	<code>\$m</code>
4	<code>\$m = \$val1</code>	3	8	3
5	<code>if (\$val2 > \$m)</code>	This evaluates to true		
6	<code>\$m = \$val2</code>	3	8	8
7	<code>return \$m</code>			

When the flow of execution reaches the end of the function, the value 8 is returned from the function to the main code and the flow of execution continues from where it was before calling the function.

Step	Statements of Main Code	\$a	\$b	\$max
8	<code>echo \$max</code>	3	8	8

Exercise 39.4-1

Back to Basics – Calculating the Sum of Two Numbers

- Do the following:
- i. Write a function named `sum()` that accepts two numeric values through its formal argument list and then calculates and returns their sum.
 - ii. Using the function `sum()` cited above, write a PHP script that lets the user enter two numbers and then displays their sum. Next, create a trace table to determine the values of the variables in each step of the PHP script for two different executions.
- The input values for the two executions are: (i) 2, 4; and (ii) 10, 20.

Solution

In this exercise you need to write a function that accepts two values from the caller (this is the main code) and then calculates and returns their sum. The solution is shown here.

file_39.4-1.php

```
<?php
function sum($a, $b) {
    $s = $a + $b;
    return $s;
}

$num1 = trim(fgets(STDIN));
$num2 = trim(fgets(STDIN));

$result = sum($num1, $num2);
echo "The sum of ", $num1, " + ", $num2, " is ", $result;
?>
```

Now, let’s create the corresponding trace tables. Since you have become more experienced with them, the column “Notes” has been removed.

i. For the input values of 2, 4, the trace table looks like this.

Step	Statement	Main Code			Function sum()		
		\$num1	\$num2	\$result	\$a	\$b	\$s
1	\$num1 = trim(fgets(STDIN))	2	?	?			
2	\$num2 = trim(fgets(STDIN))	2	4	?			
3	\$result = sum(\$num1, \$num2)				2	4	?
4	\$s = \$a + \$b				2	4	6
5	return \$s	2	4	6			
6	echo "The sum of ", ...	The message “The sum of 2 + 4 is 6” is displayed.					

ii. For the input values of 10, 20, the trace table looks like this.

Step	Statement	Main Code			Function sum()		
		\$num1	\$num2	\$result	\$a	\$b	\$s
1	\$num1 = trim(fgets(STDIN))	10	?	?			
2	\$num2 = trim(fgets(STDIN))	10	20	?			
3	\$result = sum(\$num1, \$num2)				10	20	?

4	<code>\$s = \$a + \$b</code>				10	20	30
5	<code>return \$s</code>	10	20	30			
6	<code>echo "The sum of ", ...</code>	The message “The sum of 10 + 20 is 30” is displayed.					

Exercise 39.4-2

Calculating the Sum of Two Numbers Using Fewer Lines of Code!

Rewrite the PHP script of the previous exercise using fewer lines of code.

Solution

The solution is shown here.

file_39.4-2.php

```
<?php
function sum($a, $b) {
    return $a + $b;
}

$num1 = trim(fgets(STDIN));
$num2 = trim(fgets(STDIN));

echo "The sum of ", $num1, " + ", $num2, " is ", sum($num1, $num2);
?>
```

Contrary to the solution of the previous exercise, in this function `sum()` the sum is not assigned to variable `$s` but it is directly calculated and returned. Furthermore, this main code doesn't assign the returned value to a variable but directly displays it.

Notice: User-defined functions can be called just like the built-in functions of PHP.

39.5 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

- There are two categories of functions in PHP.
- The variables that are used to pass values to a function are called arguments.
- The function `trim()` is a user-defined function.
- Every call to a user-defined function is made in the same way as a call to the built-in functions of PHP.
- There can be as many arguments as you wish in a function's formal argument list.
- In a function, the formal argument list must contain at least one argument.
- In a function, the formal argument list is optional.

8. A function cannot return an array.

9. The statement

```
return $x + 1;
```

is a valid PHP statement.

10. A formal argument can be an expression.

11. An actual argument can be an expression.

12. A function can have no arguments in the actual argument list.

13. The next statement calls the function `cube_root()` three times.

```
$cb = cube_root($x) + cube_root($x) / 2 + cube_root($x) / 3;
```

14. The following code fragment

```
$cb = cube_root($x);  
$y = $cb + 5;  
echo $y;
```

displays exactly the same value as the statement `echo cube_root($x) + 5`.

15. A function must always include a `return` statement.

16. The name `play-the-guitar` can be a valid function name.

17. You can place your functions either above or below your main code.

18. When the main code calls a function, the execution of the statements of the main code is interrupted.

19. Generally speaking, it is possible for a function to return no values to the caller.

20. The function `strpos()` is a built-in function of PHP.

21. The following code fragment

```
function add($a, $b) {  
    return $a / $b;  
}  
  
$a = 10;  
$b = 5;  
echo add($b, $a);
```

displays the value 0.5.

22. The following code fragment

```
$y = cube_root($x);  
$y += 5;
```

is equivalent to the statement `$y = 5 + cube_root($x);`

39.6 Review Exercises

Complete the following exercises.

1. The following function contains a logic error. Can you spot it?

```
function find_max($a, $b) {  
    if ($a > $b) {  
        $max = $a;  
    }  
    else {  
        $max = $b;  
    }  
}
```

2. Create a trace table to determine the values of the variables in each step of the following PHP script.

```
<?php  
function sum_digits($a) {  
    $d1 = $a % 10;  
    $d2 = intval($a / 10);  
  
    return $d1 + $d2;  
}  
  
$s = 0;  
for ($i = 25; $i <= 27; $i++) {  
    $s += sum_digits($i);  
}  
echo $s;  
?>
```

3. Create a trace table to determine the values of the variables in each step of the following PHP script.

```
<?php  
function sss($a) {  
    $sum = 0;  
    for ($k = 1; $k <= $a; $k++) {  
        $sum += $k;  
    }  
    return $sum;  
}  
  
$i = 1;
```

```
$s = 0;
while($i < 6) {
    if ($i % 2 == 1) {
        $s += 1;
    }
    else {
        $s += sss($i);
    }
    $i++;
}
echo $s;
?>
```

4. Create a trace table to determine the values of the variables in each step of the following PHP script when the value 12 is entered.

```
<?php
function custom_div($b, $d) {
    return intval(($b + $d) / 2);
}

$k = trim(fgets(STDIN));
$m = 2;
$a = 1;
while ($a < 6) {
    if ($k % $m != 0) {
        $x = custom_div($a, $m);
    }
    else {
        $x = $a + $m + custom_div($m, $a);
    }
    echo $m, " ", $a, " ", $x;
    $a += 2;
    $m++;
}
?>
```

5. Write a function named `my_round()` that accepts a real through its formal argument list and returns it rounded to two decimal places. Try not to use the `round()` function of PHP.
6. Do the following:
- Write a function named `find_min()` that accepts two numbers through its formal argument list and returns the lowest one.

- ii. Using the function `find_min()` cited above, write a PHP script that prompts the user to enter four numbers and then displays the lowest one.

7. Do the following:

- i. Write a function named `Kelvin_to_Fahrenheit()` that accepts a temperature in degrees Kelvin through its formal argument list and returns its degrees Fahrenheit equivalent.
- ii. Write a function named `Kelvin_to_Celsius()` that accepts a temperature in degrees Kelvin through its formal argument list and returns its degrees Celsius equivalent.
- iii. Using the functions `Kelvin_to_Fahrenheit()` and `Kelvin_to_Celsius()` cited above, write a PHP script that prompts the user to enter a temperature in degrees Kelvin and then displays its degrees Fahrenheit and its degrees Celsius equivalent.

It is given that

$$Kelvin = \frac{Fahrenheit + 459.67}{1.8}$$

and

$$Kelvin = Celsius + 273.15$$

- 8. The Body Mass Index (BMI) is often used to determine whether a person is overweight or underweight for his or her height. The formula used to calculate the BMI is

$$BMI = \frac{weight \cdot 703}{height^2}$$

Do the following:

- i. Write a function named `bmi()` that accepts a weight and a height through its formal argument list and then returns an action (a message) according to the following table.

BMI	Action
BMI < 16	You must add weight.
16 ≤ BMI < 18.5	You should add some weight.
18.5 ≤ BMI < 25	Maintain your weight.
25 ≤ BMI < 30	You should lose some weight.
30 ≤ BMI	You must lose weight.

- ii. Using the function `bmi()` cited above, write a PHP script that prompts the user to enter his or her weight (in pounds), age (in years), and height (in inches), and then calculates and displays the corresponding message. Moreover, the script should validate data input and display an error message when the user enters

- a. any non-numeric or negative value for weight
- b. any non-numeric or negative value for height
- c. any non-numeric, or a value less than 18 for age

Chapter 40

User-Defined Procedures

40.1 Writing your Own Procedures in PHP

The only difference between a procedure and a function is that a procedure does not directly return a result. As already stated, PHP—unlike other computer languages such as Visual Basic and FORTRAN—does not support procedures. However, there is a form of PHP function that returns no result and can emulate the way a procedure operates. You could say that PHP, to some extent, supports procedures indirectly!

On the other hand, you need to be very careful when talking to other people about PHP! Don't ever say to anyone that there are procedures in PHP—he or she will certainly laugh at you! In PHP, there are only functions! In some cases, however, and only within this book, some of them will be referred to as procedures. The reason for all this trouble is that this book's goal is to teach you “Algorithmic Thinking,” and not just the special statements that only PHP supports. Thus, it is critical for you to learn about the theory of procedures separately from functions, since there is always a chance that you may in the future deal with a computer language that actually does support procedures! So, let's say that this is a convention used only in this book!

The general form of a PHP function that actually behaves like a procedure is

```
function name([arg1, arg2, arg3, ...]) {  
    A statement or block of statements  
}
```

where

- *name* is the name of the procedure. It follows the same rules as those used for variable names.
- *arg1, arg2, arg3, ...* is a list of arguments (variables) used to pass values from the caller to the procedure. There can be as many arguments as you want.

Notice: Please note that arguments are optional; that is, a procedure may contain no arguments.

Remember! The difference between a function and a procedure is that a function returns a result, whereas a procedure doesn't.

For example, the next procedure calculates the sum of two numbers and displays the result.

```
function display_sum($num1, $num2) {  
    $result = $num1 + $num2;  
    echo $result;  
}
```

Remember! In computer science, a procedure returns no results! In PHP there are no real procedures but only functions that can behave like procedures!

Notice: In C, as well as in other similar computer languages such as C++, C#, and Java, a function that returns no result is known as a void function.

40.2 How Do You Call a Procedure?

You can call a procedure by just writing its name. The next example creates the procedure `display_line()` and the main code calls the procedure whenever it needs to display a horizontal line.

file_40.2a.php

```
<?php
function display_line() {
    echo "-----";
}

echo "Hello there!\n";
display_line();
echo "How do you do?\n";
display_line();
echo "What is your name?\n";
display_line();
?>
```

You can also pass (send) values to a procedure, as long as at least one argument exists in procedure's parenthesis. In the next example, the procedure `display_line()` is called three times but each time a different value is passed through the variable `$length`, resulting in three printed lines of different length.

file_40.2b.php

```
<?php
function display_line($length) {
    for ($i = 1; $i <= $length; $i++) {
        echo "-";
    }
    echo "\n";
}

echo "Hello there!\n";
display_line(12);
echo "How do you do?\n";
display_line(14);
echo "What is your name?\n";
display_line(18);
?>
```

Remember! Since a procedure returns no value, you **cannot** assign the returned value to a variable. The following line of code is **wrong**.

```
$y = display_line(12);
```

Also, you **cannot** call a procedure within a statement. The following line of code is also **wrong**.

```
echo "Hello there!\n", display_line(12);
```

40.3 Formal and Actual Procedure Arguments

Each procedure contains an argument list called a formal argument list. As already stated, arguments in this list are optional. The formal argument list may contain no arguments, one argument, or more than one argument.

When the procedure is called, an argument list is passed to the procedure. This list is called the actual argument list.

In the next example, the variables `$number1`, `$number2`, `$number3`, and `$number4` constitute the formal argument list whereas variables `$a`, `$b`, `$c`, and `$d` constitute the actual argument list.

file_40.3.php

```
<?php
function add_and_display ($number1, $number2, $number3, $number4) {

    $result = $number1 + $number2 + $number3 + $number4;
    echo $result;
}

$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));
$d = trim(fgets(STDIN));

add_and_display ($a, $b, $c, $d);
?>
```

This is the formal argument list

This is the actual argument list

Remember! There is a one-to-one match between the formal and the actual arguments. The value of argument `$a` is passed to argument `$number1`, the value of argument `$b` is passed to argument `$number2`, and so on.

40.4 How Does a Procedure Execute?

When the main code calls a procedure, the following steps are performed:

- The execution of the statements of the main code is interrupted.
- The values of the variables or the result of the expressions that exist in the actual argument list are passed (assigned) to the corresponding arguments (variables) in the formal argument list and the flow of execution goes to where the procedure is written.
- The statements of the procedure are executed.
- When the flow of execution reaches the end of the procedure, the flow of execution continues from where it was before calling the procedure.

In the next PHP script, the procedure `minimum()` accepts three arguments (numeric values) through its formal argument list and displays the lowest value.

file_40.4.php

```
<?php
function minimum($val1, $val2, $val3) {
    $min = $val1;
    if ($val2 < $min) {
        $min = $val2;
    }
    if ($val3 < $min) {
        $min = $val3;
    }
    echo $min;
}

$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));

minimum($a, $b, $c);

echo "The end";

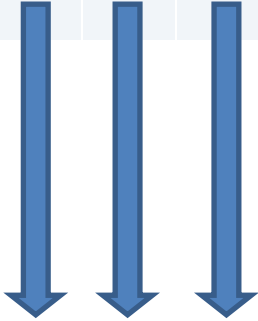
?>
```

When the PHP script starts running, the first statement executed is the statement `$a = trim(fgets(STDIN))` (this is considered the first statement of the script). Suppose the user enters the values 9, 6, and 8.

Step	Statements of Main Code	\$a	\$b	\$c
1	<code>\$a = trim(fgets(STDIN))</code>	9	?	?
2	<code>\$b = trim(fgets(STDIN))</code>	9	6	?

3	\$c = trim(fgets(STDIN))	9	6	8
4	minimum(\$a, \$b, \$c)			

When a call to the procedure `minimum()` is made, the execution of the statements of the main code is interrupted and the values of the variables `$a`, `$b`, and `$c` are copied (assigned) to the corresponding arguments (variables) `$val1`, `$val2`, and `$val3`, and the flow of execution goes to where the procedure is written. Then the statements of the procedure are executed.



Step	Statements of Procedure <code>minimum()</code>	<code>\$val1</code>	<code>\$val2</code>	<code>\$val3</code>	<code>\$min</code>
5	<code>\$min = \$val1</code>	9	6	8	9
6	<code>if (\$val2 < \$min)</code>	This evaluates to true			
7	<code>\$min = \$val2</code>	9	6	8	6
8	<code>if (\$val3 < \$min)</code>	This evaluates to false			
9	<code>echo \$min</code>	Value 6 is displayed			

When the flow of execution reaches the end of the procedure the flow of execution simply continues from where it was before calling the procedure.

Step	Statements of Main Code	<code>\$a</code>	<code>\$b</code>	<code>\$c</code>
10	<code>echo "The end"</code>	The message "The end" is displayed.		

Notice: Please note that at step 10 no values are returned from the procedure to the main code.

Remember! In PHP there are no real procedures. There are only functions that can behave like procedures and return no result. This type of function, in some computer languages like C, C++, C#, and Java are known as void functions.

Exercise 40.4-1

Back to Basics – Displaying the Absolute Value of a Number

Do the following:

- i. Write a procedure named `display_abs()` that accepts a numeric value through its formal argument list and then displays its absolute value. Do not use the built-in `abs()` function of PHP.
- ii. Using the procedure `display_abs()` cited above, write a PHP script that lets the user enter a number and then displays its absolute value followed by the initial

value given. Next, create a trace table to determine the values of the variables in each step of the PHP script for two different executions. The input values for the two executions are: (i) 5, and (ii) -5.

Solution

In this exercise you need to write a procedure that accepts a value from the caller (this is the main code) and then calculates and displays its absolute value. The solution is shown here.

file_40.4-1.php

```
<?php
function display_abs($n) {
    if ($n < 0) {
        $n = (-1) * $n;
    }
    echo $n;
}

$a = trim(fgets(STDIN));
display_abs($a);      //This displays the absolute value of given number.
echo $a;              //This displays the initial number given.
?>
```

Now, let’s create the corresponding trace tables. As already stated, since you have become more experienced with trace tables, the column “Notes” has been removed.

i. For the input value of 5, the trace table looks like this.

Step	Statement	Main Code	Procedure display_abs()
		\$a	\$n
1	\$a = trim(fgets(STDIN))	5	
2	display_abs(\$a)		5
3	if (\$n < 0)	This evaluates to false	
4	echo \$n	Value 5 is displayed	
5	echo \$a	Value 5 is displayed	

ii. For the input value of -5, the trace table looks like this.

Step	Statement	Main Code	Procedure display_abs()
		\$a	\$n
1	\$a = trim(fgets(STDIN))	-5	

2	display_abs(\$a)		-5
3	if (\$n < 0)	This evaluates to true	
4	\$n = (-1) * \$n		5
5	echo \$n	Value 5 is displayed	
6	echo \$a	Value -5 is displayed	

Notice: Please note that at step 5 the variable `$n` of the procedure contains the value 5 but when the flow of execution returns to the main code at step 6, the variable `$a` of the main code contains the value -5. Actually, the value of variable `$a` of the main code had never changed!

Exercise 40.4-2 A Simple Currency Converter

Do the following:

- i. Write a procedure named `display_menu()` that displays the following menu.
 - 1. Convert USD to Euro (EUR)
 - 2. Convert Euro (EUR) to USD
 - 3. Exit
- ii. Using the procedure `display_menu()` cited above, write a PHP script that displays the previously mentioned menu and prompts the user to enter a choice (of 1, 2, or 3) and an amount of money. The script should then calculate and display the required value. The process should repeat as many times as the user wishes. It is given that $\$1 = 0.72 \text{ EUR (€)}$.

Solution

The solution is very simple and needs no further explanation.

file_40.4-2.php

```
<?php
function display_menu() {
    echo "1. Convert USD to Euro (EUR)\n";
    echo "2. Convert Euro (EUR) to USD\n";
    echo "3. Exit\n";
    echo "-----\n";
    echo "Enter a choice: ";
}

do {
    display_menu();
    $choice = trim(fgets(STDIN));
```

```
if ($choice == 3) {
    echo "Bye!";
}
else {
    echo "Enter an amount: ";
    $amount = trim(fgets(STDIN));
    if ($choice == 1) {
        echo $amount, " USD = ", $amount * 0.72, " Euro\n";
    }
    else {
        echo $amount, " Euro = ", $amount / 0.72, " USD\n";
    }
}
} while ($choice != 3);
?>
```

40.5 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. In Java, a subprogram that returns no result is known as a void function.
2. In PHP, you can call a procedure just by writing its name.
3. In a procedure call that is made in main code, within the actual argument list there must be only variables of the main code.
4. In a procedure, all formal arguments must have different names.
5. An actual argument can be an expression.
6. A procedure must always include at least one argument in its formal argument list.
7. There is a one-to-one match between the formal and the actual arguments.
8. You can call a procedure within a statement.
9. When the flow of execution reaches the end of a procedure, the flow of execution continues from where it was before calling the procedure.
10. It is possible for a procedure to return no values to the caller.
11. It is possible for a procedure to accept no values from the caller.
12. A call to a procedure is made differently from a call to a function.
13. In the following code fragment

```
function message() {
    echo "Hello Aphrodite!";
}
```



```
echo "Hi there!";  
message();
```

the first statement that executes is the statement `echo "Hello Aphrodite!"`

40.6 Review Exercises

Complete the following exercises.

1. Create a trace table to determine the values of the variables in each step of the following PHP script when the values 3, 7, 9, 2, and 4 are entered.

```
<?php  
function display($a) {  
    if ($a % 2 == 0) {  
        echo $a, " is even";  
    }  
    else {  
        echo $a, " is odd";  
    }  
}  
  
for ($i = 1; $i <= 5; $i++) {  
    $x = trim(fgets(STDIN));  
    display($x);  
}  
?>
```

2. Create a trace table to determine the values of the variables in each step of the following PHP script.

```
<?php  
function division($a, $b) {  
    $b = intval($b / $a);  
    echo $a * $b, "\n";  
}  
  
$x = 20;  
$y = 30;  
while ($x % $y < 30) {  
    division($y, $x);  
    $x = 4 * $y;  
    $y++;  
}  
?>
```

3. Create a trace table to determine the values of the variables in each step of the following PHP script when the values 2, 3, and 4 are entered.

```
<?php
function calculate($n) {
    $s = 0;
    for ($j = 2; $j <= 2 * $n; $j += 2) {
        $s = $s + pow($j, 2);
    }

    echo $s;
}

for ($i = 1; $i <= 3; $i++) {
    $m = trim(fgets(STDIN));
    calculate($m);
}
?>
```

4. Write a procedure that accepts five values through its formal argument list and then displays the greatest value.
5. Do the following:
- Write a procedure named `num_of_days()` that accepts a year and a month (1 – 12) through its formal argument list and then displays the number of days in that month. Take special care when a year is a leap year; that is, a year in which February has 29 instead of 28 days.
Hint: A year is a leap year when it is exactly divisible by 4 and not by 100, or when it is exactly divisible by 400.
 - Using the procedure `num_of_days()` cited above, write a PHP script that prompts the user to enter a year and then displays the number of the days in each month of that year.
6. Do the following:
- Write a procedure named `display_menu()` that displays the following menu.
 - Convert meters to miles
 - Convert miles to meters
 - Exit
 - Write a procedure named `meters_to_miles()` that accepts a value in meters through its formal argument list and then displays the message “XX meters equals YY miles” where XX and YY must be replaced by actual values.

- iii. Write a procedure named `miles_to_meters()` that accepts a value in miles through its formal argument list and then displays the message “YY miles equals XX meters” where XX and YY must be replaced by actual values.
 - iv. Using the procedures `meters_to_miles()` and `miles_to_meters()` cited above, write a PHP script that displays the previously mentioned menu and prompts the user to enter a choice (of 1, 2, or 3) and a distance. The script should then calculate and display the required value. The process should repeat as many times as the user wishes. It is given that 1 mile = 1609.344 meters.
7. The LAV Cell Phone Company charges customers a basic rate of \$10 per month, and additional rates are charged based on the total number of seconds a customer talks on his or her cell phone within the month. Use the rates shown in the following table.

Number of Seconds a Customer Talks on his or her Cell Phone	Additional Rates (in USD per second)
1 – 600	Free of charge
601 – 1200	\$0.01
1201 and above	\$0.02

Do the following:

- i. Write a procedure named `amount_to_pay()` that accepts a number in seconds through its formal argument list and then displays the total amount to pay. Please note that the rates are progressive. Moreover, federal, state, and local taxes add a total of 11% to each bill
- ii. Using the procedure `amount_to_pay()` cited above, write a PHP script that prompts the user to enter the number of seconds he or she talks on the cell phone and then displays the total amount to pay.

Chapter 41

Tips and Tricks with Subprograms

41.1 Can Two Subprograms use Variables of the Same Name?

You should realize something: each subprogram uses its own memory space to hold the values of its variables. Even the main code has its own memory space! This means that you can have a variable named `$test` in main code, another variable named `$test` in a subprogram, and yet another variable named `$test` in another subprogram. Pay attention! Those three variables are three completely different variables, in different memory locations, and they can hold completely different values.

The trace table in the following example can help you understand what really goes on.

file_41.1a.php

```
<?php
function f1() {
    $test = 22;
    echo $test;
    return true;
}
function f2($test) {
    echo $test;
}

$test = 5;
echo $test;
$ret = f1();
f2(10);
echo $test;
?>
```

The trace table is shown here.

Step	Statement	Notes	Main Code		Function f1 ()	Procedure f2 ()
			\$test	\$ret	\$test	\$test
1	\$test = 5		5	?		
2	echo \$test	Value 5 is displayed	5	?		
3	\$ret = f1()	f1() is called			?	
4	\$test = 22				22	
5	echo \$test	Value 22 is displayed			22	
6	return true	It returns to the main code	5	true		

7	f2(10)	f2() is called				10
8	echo \$test	Value 10 is displayed				10
9	echo \$test	Value 5 is displayed	5	true		

As you can see in the trace table, there are three variables named `$test` in three different memory locations and each one of them holds a completely different value. Now, let's see something else. In the next PHP script, the variable `$test` of the main code is passed to function `f1()` through an argument that also happens to be a variable named `$test`. As already stated, even though both variables have the same name, they are actually two different variables in two different locations in main memory! In reality, this means that although `f1()` alters the value of its variable `$test`, when the flow of execution returns to the main code, this change does not affect the value of the variable `$test` of the main code.

file_41.1b.php

```
<?php
function f1($test) {
    $test++;
    echo $test; //This is the variable of procedure f1().
               //Value 6 is displayed
}

$test = 5;
f1($test);
echo $test;   //This is the variable of the main code.
               //Value 5 is displayed
?>
```

Notice: Please note that variables used in a subprogram “live” as long as the subprogram is being executed. This means that before calling the subprogram, none of its variables (including those in the formal argument list) exists in main memory (RAM). They are all created in the main memory when the subprogram is called, and they are all removed from the main memory when the subprogram finishes and the flow of execution returns to the caller. The only variables that “live” forever, or at least for as long as the PHP script is being executed, are the variables of the main code!

41.2 Can a Subprogram Call Another Subprogram?

All the time you’ve been reading this section, you may have had the impression that only the main code can call a subprogram. Of course, this is not true! A subprogram can call any other subprogram which in turn can call another subprogram, and so on. You can make whichever combination you wish. For example, you can write a function that calls a procedure, a procedure that calls a

function, a function that calls another function, or even a function that calls one of the built-in functions of PHP.

The next example presents exactly this situation. The main code calls the procedure `display_sum()`, which in turn calls the function `add()`.

file_41.2.php

```
<?php
function add($number1, $number2) {
    $result = $number1 + $number2;
    return $result;
}

function display_sum($num1, $num2) {
    echo add($num1, $num2);
}

$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));

display_sum($a, $b);
?>
```

Notice: Please note that there is no restriction on the order in which the two subprograms should be written. It would have been exactly the same if the procedure `display_sum()` had been written before the function `add()`.

Exercise 41.2-1 *A Currency Converter – Using Functions with Procedures*

Do the following:

- i. Write a subprogram named `display_menu()` that displays the following menu.
 1. Convert USD to Euro (EUR)
 2. Convert USD to British Pound Sterling (GBP)
 3. Convert USD to Japanese Yen (JPY)
 4. Convert USD to Canadian Dollar (CAD)
 5. Exit
- ii. Write four different subprograms named `USD_to_Euro()`, `USD_to_GBP()`, `USD_to_JPY()`, and `USD_to_CAD()`, that accept a currency through their formal argument list and then return the corresponding converted value.
- iii. Using the five subprograms cited above, write a PHP script that displays the menu previously mentioned and then prompts the user to enter a choice (of 1, 2, 3, 4, or 5) and an amount in US dollars. The script should then calculate and

display the required value. The process should repeat as many times as the user wishes. It is given that

- $\$1 = 0.72 \text{ EUR (€)}$
- $\$1 = 0.60 \text{ GBP (£)}$
- $\$1 = ¥ 102.15 \text{ JPY}$
- $\$1 = 1.10 \text{ CAD (\$)}$

Solution

This solution combines the use of both functions and procedures. The four functions that convert currencies accept a value through an argument and then they return the corresponding value. The solution is shown here.

file_41.2-1.php

```
<?php
```

```
function display_menu() {  
    echo "1. Convert USD to Euro (EUR)\n";  
    echo "2. Convert USD to British Pound Sterling (GBP)\n";  
    echo "3. Convert USD to Japanese Yen (JPY)\n";  
    echo "4. Convert USD to Canadian Dollar (CAD)\n";  
    echo "5. Exit\n";  
    echo "-----\n";  
    echo "Enter a choice: ";  
}
```

```
function USD_to_Euro($value) {  
    return $value * 0.72;  
}
```

```
function USD_to_GBP($value) {  
    return $value * 0.60;  
}
```

```
function USD_to_JPY($value) {  
    return $value * 102.15;  
}
```

```
function USD_to_CAD($value) {  
    return $value * 1.10;  
}
```

```
do {
```



```
display_menu();
$choice = trim(fgets(STDIN));

if ($choice == 5) {
    echo "Bye!";
}
else {
    echo "Enter an amount in US dollars: ";
    $amount = trim(fgets(STDIN));
    switch ($choice) {
        case 1:
            echo $amount, " USD = ", USD_to_Euro($amount), " Euro\n";
            break;
        case 2:
            echo $amount, " USD = ", USD_to_GBP($amount), " GBP\n";
            break;
        case 3:
            echo $amount, " USD = ", USD_to_JPY($amount), " JPY\n";
            break;
        case 4:
            echo $amount, " USD = ", USD_to_CAD($amount), " CAD\n";
            break;
    }
}
} while ($choice != 5);
?>
```

41.3 Passing Arguments by Value and by Reference

By default, variables are passed to subprograms by value. This means that if the value of an argument is changed within the subprogram, it does not get changed outside of it. Take a look at the following example.

file_41.3a.php

```
<?php
function f1($b) {
    $b++;
    echo $b;      //Value 11 is displayed
}

$a = 10;
f1($a);
```

```
echo $a;           //Value 10 is displayed
?>
```

The value 10 of variable `$a` is passed to procedure `f1()` through argument `$b`. However, although the content of variable `$b` is altered within the procedure, when the flow of execution returns to the main code this change does not affect the value of variable `$a`. This is regardless of the fact that the main code and the procedure are using two variables with different names. The same would have happened if, for example, both the main code and the procedure had used two variables of the same name. The next PHP script operates exactly the same way and displays exactly the same results as previous script did.

file_41.3b.php

```
<?php
function f1($a) {
    $a++;
    echo $a; //Value 11 is displayed
}

$a = 10;
f1($a);
echo $a;    //Value 10 is displayed
?>
```

Still, if you want a subprogram to change the value of its arguments and also reflect this change outside of the subprogram, you must pass the arguments by reference. To do so, you must prepend an ampersand (`&`) at the beginning of the argument name in the formal argument list. Take a look at the following example, in which an ampersand (`&`) is prepended to the argument `$b` in the formal argument list.

file_41.3c.php

```
<?php
function f1(&$b) {
    $b++;
    echo $b; //Value 11 is displayed
}

$a = 10;
f1($a);
echo $a;    //Value 11 is displayed
?>
```

Notice: Please note that the ampersand (`&`) symbol is prepended only to arguments within the formal argument list. Prepending an ampersand to arguments within the actual argument list is syntactically incorrect!

Be careful about one thing though: only variables can be passed by reference. You cannot pass by reference a constant value or an expression, as shown here.

file_41.3d.php

```
<?php
function f1(&$b) {
    $b++;
}

$x = 9;
f1($x);           //This works fine!

f1(10);           //This produces an error!

f1($x + 1);       //This produces also an error!
?>
```

The second and third calls to the function produce an error. This happens because when the flow of execution returns from the procedure to the main code, the value of variable `$b` must be assigned back to an argument in the actual argument list. Obviously, this cannot be done when the argument is either a constant value or an expression.

So, as you have probably realized, passing values by reference can provide an indirect way for a subprogram to “return” values. In the next example, the function `div_mod()` divides variable `$a` by variable `$b` and finds their integer quotient and their integer remainder. If all goes well, it returns `true`; otherwise, it returns `false`. Moreover, through the two arguments `$int_quotient` and `$int_remainder`, the function also indirectly returns the calculated quotient and the calculated remainder.

file_41.3e.php

```
<?php
function div_mod($a, $b, &$int_quotient, &$int_remainder) {
    $return_value = true;
    if ($b == 0) {
        $return_value = false;
    }
    else {
        $int_quotient = intval($a / $b);
        $int_remainder = $a % $b;
    }
    return $return_value;
}

$val1 = trim(fgets(STDIN));
```

```
$val2 = trim(fgets(STDIN));  
$ret = div_mod($val1, $val2, $int_q, $int_r);  
if ($ret == true) {  
    echo $int_q, $int_r;  
}  
else {  
    echo "Sorry, wrong values entered!";  
}  
?>
```

Notice: A very good tactic regarding the arguments in the formal argument list is to have all of those being passed by value written before those being passed by reference.

Exercise 41.3-1 Finding the Logic Error

The following PHP script is supposed to prompt the user to enter an integer and then display the product of that integer multiplied by its number of digits. For example, if the user enters the value 401, the script should display a result of $401 \times 3 = 1203$. Unfortunately, the PHP script displays 0. Can you find out why?

file_41.3-1.php

```
<?php  
function get_num_of_digits(&$x) {  
    $count = 0;  
    while ($x != 0) {  
        $count++;  
        $x = intval($x / 10);  
    }  
    return $count;  
}  
  
$val = trim(fgets(STDIN));  
  
echo get_num_of_digits($val) * $val;  
?>
```

Solution

The function `get_num_of_digits()` counts the number of digits of the argument `$x` using an already known method proposed in Chapter 31. But before you read the answer below, can you try to find the logic error by yourself?

No? Come on, it isn't that difficult!

Still No? Do you want to give up?

So, let it be!

The problem is that, within the function `get_num_of_digits()`, variable `$x` eventually becomes 0, and since the variable `$val` is passed to the function by reference, that 0 also reflects back to the main code. So, when the flow of execution returns to the main code, the value of variable `$val` is zeroed!

To resolve this issue, all you have to do is remove the ampersand (`&`) sign at the beginning of the argument `$x`. If you do so, the variable `$val` is passed to the function by value, so that no matter what happens to variable `$x` within the function, nothing can affect the value of the variable `$val` of the main code.

Exercise 41.3-2 *Finding the Absolute Value of a Number*

Do the following:

- i. Write a subprogram named `my_abs()` that accepts a numeric value through its formal argument list and then calculates and returns its absolute value. Do not use the built-in `abs()` function of PHP.
- ii. Using the subprogram `my_abs()` cited above, write a PHP script that finds and displays the value of `y` in the following formula.

$$y = \frac{|5 + x|}{9} + \frac{|x + 9|}{8}$$

Solution

When writing a subprogram, there is always the same question: *which type of subprogram is most appropriate, a function or a procedure?*

As you already know, a function can return a value; but on the other hand, a procedure can also return a value (indirectly) through an argument passed by reference. So which is the most appropriate? The answer is simple: it depends on the given problem!

Let's try to solve this exercise by writing and using both a function and a procedure!

First approach – Finding the absolute value using a user-defined function.

This approach is quite simple and needs no further explanation. The solution is presented here.

`file_41.3-2a.php`

```
<?php
function my_abs($n) {
    $return_value = $n;
    if ($return_value < 0) {
        $return_value = (-1) * $return_value;
    }
    return $return_value;
}
```

```
$x = trim(fgets(STDIN));
$y = my_abs(5 + $x) / 9 + my_abs($x + 9) / 8;
echo $y;
?>
```

Notice: Please note that the function `my_abs()` is called twice within the statement that assigns the final value to variable `$y`.

Second approach – Finding the absolute value using a user-defined procedure.

If you want to use a procedure, things become complicated. Since procedures return no values, you must pass an argument by reference, giving the procedure the ability to return the result indirectly. But there is one little problem! As already stated, only variables can be passed by reference!

One solution that overcomes this problem is to use auxiliary variables, as shown here.

file_41.3-2b.php

```
<?php
function my_abs(&$n) {
    if ($n < 0) {
        $n = (-1) * $n;
    }
}

$x = trim(fgets(STDIN));

$aux1 = 5 + $x;
my_abs($aux1);

$aux2 = $x + 9;
my_abs($aux2);

$y = $aux1 / 9 + $aux2 / 8;
echo $y;
?>
```

You cannot pass an expression by reference. This is why the expression is first assigned to a variable.

Another solution would be to have two different arguments, one for passing the value and one for returning the result, as shown here.

file_41.3-2c.php

```
<?php
function my_abs($n, &$ret) {
    $ret = $n;
```

```

    if ($ret < 0) {
        $ret = (-1) * $ret;
    }
}

$x = trim(fgets(STDIN));

my_abs(5 + $x, $aux1);
my_abs($x + 9, $aux2);

$y = $aux1 / 9 + $aux2 / 8;

echo $y;

?>

```

The expressions are passed by value. Only variables `$aux1` and `$aux2` are passed by reference.

In conclusion, in this particular exercise, the first approach (the one that uses a function) is obviously more convenient. So keep one simple rule in your mind: *If a subprogram returns only one value, write a function!*

Exercise 41.3-3 Converting a Function to a Procedure

Convert the following function into a procedure.

```

<?php
function is_odd($number) {
    $return_value = true;
    if ($number % 2 == 0) {
        $return_value = false;
    }

    return $return_value;
}

?>

```

Then, using the new procedure, write a short PHP script that prompts the user to enter an integer and then displays a message indicating whether or not the result of the given integer multiplied by 3 is an odd number.

Solution

Since a procedure returns no results directly, you must add an extra argument to the formal argument list. This argument has to be passed by reference so that the value can be returned (indirectly) back to the caller. The procedure, and the main code that calls the procedure, are as follows.

file_41.3-3.php

```
<?php
function is_odd($number, &$return_value) {
    $return_value = true;
    if ($number % 2 == 0) {
        $return_value = false;
    }
}

echo "Enter an integer: ";
$x = trim(fgets(STDIN));

is_odd(3 * $x, $return_value);
if ($return_value == true) {
    echo "3 x ", $number, " is odd";
}
else {
    echo "3 x ", $number, " is not odd";
}

?>
```

41.4 Passing and/or Returning an Array

Passing an array to a subprogram as an argument is as easy as passing a simple variable. The next example passes array `$a` to the procedure `display()`, and the latter displays the array.

file_41.4a.php

```
<?php
define("ELEMENTS", 10);

function display($b) {
    for ($i = 0; $i <= ELEMENTS - 1; $i++) {
        echo $b[$i], "\t";
    }
}

$a = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $a[$i] = trim(fgets(STDIN));
}
```



```
display($a);  
?>
```

Of course, if you pass an array by reference to a subprogram, and that subprogram changes the value of one or more elements of the array, these changes are also reflected outside the subprogram. In the next example, the PHP script must find the three lowest values of array `$t`. To do so, the script calls and passes the array to the procedure `get_array()` through its formal argument `$a`, which in turn sorts array `$a` using the insertion sort algorithm. When the flow of execution returns to the main code, array `$t` also gets sorted. So what the main code finally does is just display the elements of the first three positions of the array.

file_41.4b.php

```
<?php
```

```
define("ELEMENTS", 10);
```

Array is passed by reference

```
function get_array(&$a) {
```

```
    for ($m = 1; $m <= ELEMENTS - 1; $m++) {
```

```
        $element = $a[$m];
```

```
        $n = $m;
```

```
        while ($n > 0 && $a[$n - 1] > $element) {
```

```
            $a[$n] = $a[$n - 1];
```

```
            $n--;
```

```
        }
```

```
        $a[$n] = $element;
```

```
    }
```

```
}
```

```
$t = array(75, 73, 78, 70, 71, 74, 72, 69, 79, 77);
```

```
get_array($t);
```

```
echo "Three lowest values are: ", $t[0], " ", $t[1], " ", $t[2], "\n";
```

```
//In this step, array $t is now sorted
```

```
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
```

```
    echo $t[$i], "\t";
```

```
}
```

```
?>
```

Notice: Please note that, since the array `$t` of the main code is passed to the procedure by reference, when the flow of execution returns to the main code, array `$t` also gets sorted.

So, one way in which a subprogram can indirectly return an array is by passing the array by reference. But there are many times when passing an array by reference can

be completely disastrous. Suppose you have the following two arrays. Array `$names` contains the names of 10 cities, and array `$t` contains their corresponding temperatures recorded at 12:00 p.m. on a specific day.

<code>\$names =</code>		<code>\$t =</code>	0	
			1	
			2	
			:	
			9	

Now, suppose that for array `$t` you wish to display the three lowest temperatures. If you call procedure `get_array()` of the previous PHP script, you have a problem. Although the three lowest temperatures can be displayed as required, the array `$t` becomes sorted; therefore, the relationship between its elements and the elements of array `$names` is lost forever!

One possible solution would be to write a function in which the array is passed by value (not by reference) and the function would return a smaller array that contains only the three lowest values. The proposed solution is shown here.

file_41.4c.php

```
<?php
```

```
define("ELEMENTS", 10);
```

Array is passed by value

```
function get_array($a) {
```

```
    for ($m = 1; $m <= ELEMENTS - 1; $m++) {
```

```
        $element = $a[$m];
```

```
        $n = $m;
```

```
        while ($n > 0 && $a[$n - 1] > $element) {
```

```
            $a[$n] = $a[$n - 1];
```

```
            $n--;
```

```
        }
```

```
        $a[$n] = $element;
```

```
    }
```

```
    $ret_array = array($a[0], $a[1], $a[2]);
```

```
    return $ret_array;
```

```
}
```

```
$names = array("City1", "City2", "City3", "City4", "City5",  
               "City6", "City7", "City8", "City9", "City10");
```

```
$t = array(75, 73, 78, 70, 71, 74, 72, 69, 79, 77);
```

```
$low = get_array($t);
```

```
echo "Three lowest values are: ", $low[0], " ", $low[1], " ", $low[2], "\n";

//In this step, array $t is NOT sorted
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    echo $t[$i], "\t", $names[$i], "\n";
}

?>
```

Notice: Returning an array from a subprogram is as easy as returning a simple variable.

41.5 Default Argument Values (Optional Arguments)

If you assign a default value to an argument within the formal argument list, it means that if no value is passed for that argument, the default value is used. In the next example, the function `prepend_title()` prepends a title before the name. However, if no value for argument `$title` is passed, the function uses the default value “Mr.”

file_41.5a.php

```
<?php
function prepend_title($name, $title = "Mr") {
    return $title . " " . $name;
}

echo prepend_title("John King");           //It displays Mr John King
echo prepend_title("Maria Miller", "Ms");  //it displays Ms Maria Miller

?>
```

Notice: When a default value is assigned to an argument within the formal argument list, this argument is called an optional argument.

Within the formal argument list, any optional arguments should be on the right side of any non-optional arguments; to do the opposite of this would be incorrect. Consider the following example, in which the optional argument `$title` is written before the non-optional argument `$name` in the formal argument list.

file_41.5b.php

```
<?php
function prepend_title($title = "Mr", $name ) {
    return $title . " " . $name;
}

echo prepend_title("Vassilis Bouras");

?>
```

As already mentioned, there is a one-to-one match between the formal and the actual arguments. So what actually happens here is that the value “Vassilis Bouras” is passed to the argument `$title` leaving argument `$name` with no value. Obviously, PHP will complain about it by saying that an argument is missing for the function `prepend_title()`.

Notice: *The default value must be a constant value, not a variable.*

41.6 The Scope of a Variable

The scope of a variable refers to the range of effect of that variable.

In PHP, a variable can have a local or global scope. A variable declared within a subprogram has a local scope and can be accessed only from within that subprogram. On the other hand, a variable declared outside of a subprogram has a global scope and can be accessed from within **any** subprogram, as well as from the main code. To access a global variable from within a subprogram, though, you must use the PHP keyword `global`.

Let’s see some examples. The next PHP script declares a global variable `$a` as well as a local variable `$b` within the procedure. The value of the global variable `$a`, though, is accessed and displayed within the procedure.

file_41.6a.php

```
<?php
function display_values() {
    global $a;
    $b = 3;           //Local scope
    echo $a, $b;      //This displays 10  3
}

$a = 10;             //Global scope
display_values();

?>
```

If you wish to access more than one global variable from within a subprogram, you can write the keyword `global` and, next to it, all the variables separated with commas, as shown here.

```
global $a, $b, $c;
```

Be careful though! If the value of a global variable is altered within a subprogram, this change is also reflected outside of the subprogram. In the next example, the procedure `display_values()` increases the value of variable `$a` to 11, and when the flow of execution returns to the main code, variable `$a` still contains the value 11.

file_41.6b.php

```
<?php
function display_values() {
    global $a, $b;      //Permit access to global variables $a and $b
```

```
$a++;  
echo $a, $b;           //This displays 11 20  
}  
  
$a = 10;  
$b = 20;  
echo $a, $b;           //This displays 10 20  
display_values();  
echo $a, $b;           //This displays 11 20  
?>
```

The next example declares a global variable `$a`, two local variables `$a` and `$b` within the procedure `display_values()`, and another two local variables `$a` and `$b` within the procedure `display_other_values()`. Keep in mind that the global variable `$a` and the two local variables `$a` are three different variables!

file_41.6c.php

```
<?php  
function display_values() {  
    $a = 7;           //Local scope  
    $b = 3;           //Local scope  
    echo $a, $b;      //This displays 7 3  
}  
  
function display_other_values() {  
    $a = 9;           //Local scope  
    $b = 2;           //Local scope  
    echo $a, $b;      //This displays 9 2  
}  
  
$a = 10;              //Global scope  
echo $a;              //This displays 10  
display_values();  
display_other_values();  
echo $a;              //This displays 10  
?>
```

Notice: You can have variables of local scope of the same name within different subprograms, because they are recognized only by the subprogram in which they are declared.

41.7 Converting Parts of Code into Subprograms

As already mentioned, writing large programs without subdividing them into smaller subprograms results in a code called “spaghetti code”. Suppose you have a large “spaghetti code” and you wish to subdivide it into smaller subprograms. The next script is an example explaining the steps that must be followed. The parts of the script marked with a dashed rectangle must be converted into subprograms.

file_41.7a.php

```
<?php
$total_yes = 0;
$female_no = 0;
for ($i = 1; $i <= 100; $i++) {

    do {
        echo "Enter Sex for Citizen No ", $i, ": ";
        $temp1 = trim(fgets(STDIN));
        $sex = strtolower($temp1);
    } while ($sex != "male" && $sex != "female");

    do {
        echo "Do you go jogging in the afternoon? ";
        $temp2 = trim(fgets(STDIN));
        $answer = strtolower($temp2);
    } while ($answer != "yes" && $answer != "no" && $answer != "sometimes");

    if ($answer == "yes") {
        $total_yes++;
    }

    if ($sex == "female" && $answer == "no") {
        $female_no++;
    }

}

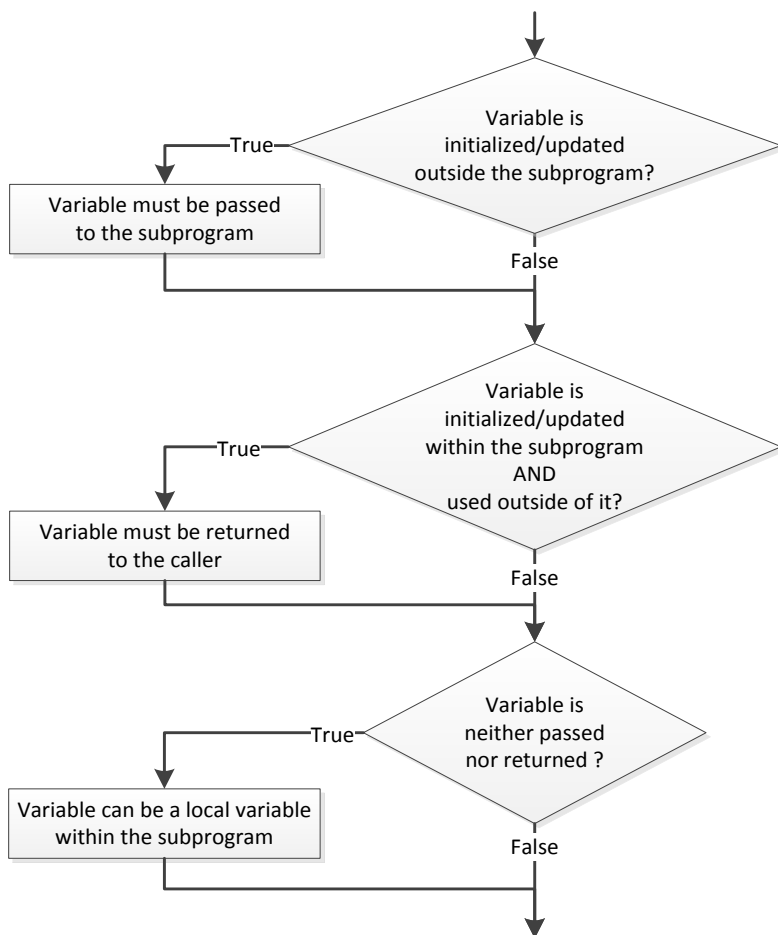
echo "Total positive answers: ", $total_yes;
echo "Women's negative answers: ", $female_no;
?>
```

To convert parts of this script into subprograms you must:

- decide, for each part, whether to use a function or a procedure. The factor that helps you decide what to use is that a function returns a result, whereas

a procedure doesn't. Of course, if you pass your arguments by reference, then both of them can return many results—indirectly, of course.

- determine which variables participate in each part of the script and their roles in that part. The flowchart that follows can help you decide what to do with each variable, whether it should be passed to the subprogram and/or returned from the subprogram, or if it should just be a local variable within the subprogram.



So, with the help of this flowchart, let's discuss each part of the script independently! The parts that are not marked with a dashed rectangle will comprise the main code.

First part

In the first part of the script, there are three variables: `$i`, `$temp1`, and `$sex`. However, not all of them must be included in the formal argument list. Let's find out why!

- Variable `$i`
 - is initialized/updated outside the subprogram; thus, it must be passed to the subprogram
 - is not updated within the subprogram; thus, it should not be returned to the caller
- Variable `$temp1`
 - is not initialized/updated outside of the subprogram; thus, it should not be passed to the subprogram
 - is initialized within the subprogram but its value is not used outside of it; thus, it should not be returned to the caller

According to the flowchart, since variable `$temp1` should neither be passed nor returned, this variable can just be a local variable within the subprogram.

- Variable `$sex`
 - is not initialized/updated outside of the subprogram; thus, it should not be passed to the subprogram
 - is initialized within the subprogram and its value is used outside of it; thus, it must be returned to the caller

Therefore, since only one value must be returned to the main code, a function can be used as shown here.

```
function part1($i) {  
  do {  
    echo "Enter Sex for Citizen No ", $i, ": ";  
    $temp1 = trim(fgets(STDIN));  
    $sex = strtolower($temp1);  
  } while ($sex != "male" && $sex != "female");  
  
  return $sex;  
}
```

Second part

In the second part of the script there are two variables, `$temp2` and `$answer`, but they do not both need to be included in the formal argument list. Let's find out why!

- Variable `$temp2`
 - is not initialized/updated outside of the subprogram; thus, it should not be passed to the subprogram
 - is initialized/updated within the subprogram but its value is not used outside of it; thus, it should not be returned to the caller

According to the flowchart, since variable `$temp2` should neither be passed nor returned, this variable can just be a local variable within the subprogram.

- **Variable \$answer**
 - is not initialized/updated outside of the subprogram; thus, it should not be passed to the subprogram
 - is initialized within the subprogram and its value is used outside of it; thus, it must be returned to the caller

Therefore, since only one value must be returned to the main code, a function can be used, as shown here.

```
function part2() {
  do {
    echo "Do you go jogging in the afternoon? ";
    $temp2 = trim(fgets(STDIN));
    $answer = strtolower($temp2);
  } while ($answer != "yes" && $answer != "no" && $answer != "sometimes");

  return $answer;
}
```

Third part

In the third part of the example, there are four variables: \$answer, \$total_yes, \$sex and \$female_no and all of them must be included in the formal argument list. Let's find out why!

- **Both variables \$answer and \$sex**
 - are initialized/updated outside of the subprogram; thus, they must be passed to the subprogram
 - are not updated within the subprogram; thus, they should not be returned to the caller
- **Both variables \$total_yes and \$female_no**
 - are initialized outside of the subprogram; thus, they must be passed to the subprogram
 - are updated within the subprogram and their value is used outside of it; thus, they must be returned to the caller

Therefore, since two values must be returned to the main code, a procedure can be used. The variables \$total_yes and \$female_no must be passed by reference so that their values can be returned to the main code (indirectly), as shown here.

```
function part3($answer, $sex, &$total_yes, &$female_no) {
  if ($answer == "yes") {
    $total_yes++;
  }

  if ($sex == "female" && $answer == "no") {
    $female_no++;
  }
}
```

```
}  
  
}
```

Remember! *A very good tactic regarding the argument in the formal argument list is to have all of those being passed by value before those being passed by reference.*

Fourth part

In the fourth part of the example, there are two variables: `$total_yes` and `$female_no`. Let's see what you should do with them.

- Both variables `$total_yes` and `$female_no`
 - are updated outside of the subprogram; thus, they must be passed to the subprogram
 - are not updated within the subprogram; thus, they should not be returned to the caller

Therefore, since no value should be returned to the main code, a procedure can be used, as shown here.

```
function part4($total_yes, $female_no) {  
    echo "Total positive answers: ", $total_yes;  
    echo "Women's negative answers: ", $female_no;  
}
```

The final script

The final script, including the main code and all the subprograms cited above, is shown here.

file_41.7b.php

```
<?php  
function part1($i) {  
    do {  
        echo "Enter Sex for Citizen No ", $i, ": ";  
        $temp1 = trim(fgets(STDIN));  
        $sex = strtolower($temp1);  
    } while ($sex != "male" && $sex != "female");  
  
    return $sex;  
}  
  
function part2() {  
    do {  
        echo "Do you go jogging in the afternoon? ";  
        $temp2 = trim(fgets(STDIN));  
        $answer = strtolower($temp2);
```

```
} while ($answer != "yes" && $answer != "no" && $answer != "sometimes");

return $answer;
}

function part3($answer, $sex, &$total_yes, &$female_no) {
    if ($answer == "yes") {
        $total_yes++;
    }

    if ($sex == "female" && $answer == "no") {
        $female_no++;
    }
}

function part4($total_yes, $female_no) {
    echo "Total positive answers: ", $total_yes;
    echo "Women's negative answers: ", $female_no;
}

$total_yes = 0;
$female_no = 0;
for ($i = 1; $i <= 100; $i++) {
    $sex = part1($i);
    $answer = part2();

    part3($answer, $sex, $total_yes, $female_no);
}

part4($total_yes, $female_no);

?>
```

41.8 Recursion

Recursion is a programming technique in which a subprogram calls itself. This might initially seem like an endless loop, but of course this is not true; a subprogram that uses recursion must be written in a way that obviously satisfies the property of finiteness.

Imagine that the next PHP script helps you find your way home. In this script, recursion occurs because the procedure `find_your_way_home()` calls itself within the procedure.

```
<?php
```

```
function find_your_way_home () {  
    if (you_are_already_at_home == true) {  
        stop_walking();  
    }  
    else {  
        take_one_step_toward_home();  
        find_your_way_home();  
    }  
}  
  
find_your_way_home();  
?>
```

Now, let's try to analyze recursion through a real example. The next PHP script calculates the factorial of 5 using recursion.

file_41.8.php

```
<?php  
function factorial($value) {  
    if ($value == 1) {  
        $return_value = 1;  
    }  
    else {  
        $return_value = $value * factorial($value - 1);  
    }  
  
    return $return_value;  
}  
  
echo factorial(5);           //This displays 120  
?>
```

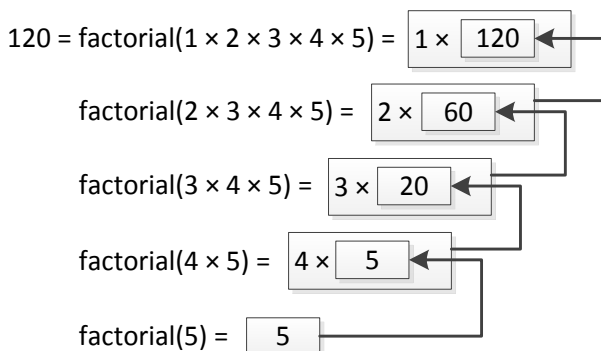
Remember! In mathematics, the factorial of a non-negative integer N is the product of all positive integers less than or equal to N . It is denoted by $N!$ and the factorial of 0 is, by definition, equal to 1. For example, the factorial of 5 is $1 \times 2 \times 3 \times 4 \times 5 = 120$.

Notice: Recursion occurs because the function `factorial()` calls itself within the function.

Notice: Please note that there isn't any loop control structure!

You are probably confused right now. How on Earth was the product $1 \times 2 \times 3 \times 4 \times 5$ calculated without using a loop control structure? The next diagram may help you understand. It shows the multiplication operations that are performed as function `factorial()` works its way backwards through the series of calls. When the function

`factorial()` returns from the topmost call, you have the final solution. Read this diagram backwards, from bottom to top.



In conclusion, all recursive subprograms must follow three important rules.

1. They must have a base case
2. They must change their state and move toward the base case
3. They must call themselves

where

- the base case is the condition that “tells” the subprogram to stop recursions. The base case is usually a very small problem that can be solved directly. In the function `factorial()`, the base case is the factorial of 1. When `factorial(1)` is called, the Boolean expression (`$value == 1`) validates to true and marks the end of the recursions.
- a change of state means that the subprogram alters some of its data. Usually, data are getting smaller and smaller in some way. In the function `factorial()`, since the base case is the factorial of 1, the whole concept relies on the idea of moving toward that base case.
- The last rule just states the obvious; the subprogram must call itself.

Exercise 41.8-1 Calculating the Fibonacci Sequence Recursively

The Fibonacci sequence is a series of numbers in the following sequence:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

By definition, the first two numbers are 1 and 1 and each subsequent number is the sum of the previous two.

Write a PHP script that lets the user enter a positive integer N and then calculates recursively and finally displays the N^{th} term of the Fibonacci series.

Solution

In the following PHP script, the function `Fib()` calculates the N^{th} term of the Fibonacci series recursively. This is probably not the best algorithm in the world. It is

mentioned here, however, just to show you how recursion can sometimes make things worse!

file_41.8-1.php

```
<?php
function Fib($n) {
    if ($n == 0 || $n == 1) {
        $return_value = $n;
    }
    else {
        $return_value = Fib($n - 1) + Fib($n - 2);
    }

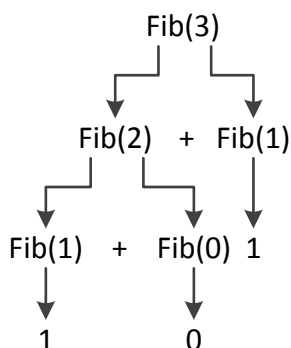
    return $return_value;
}

$num = trim(fgets(STDIN));
echo Fib($num);
?>
```

To explain how this function works, the best thing to do is use some input values.

For input value 3

If the user enters a value of 3, the following recursive calls are made.



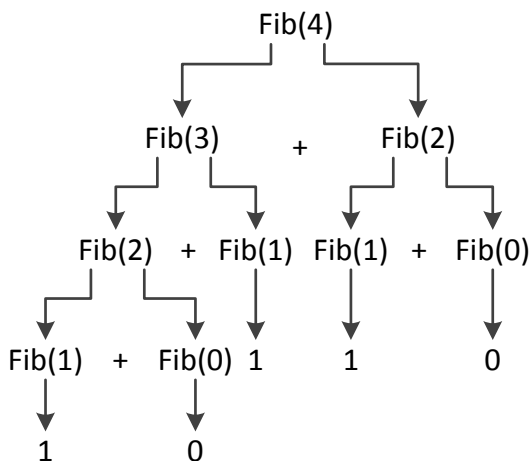
If you calculate the results of these recursive calls, you have

$$(1 + 0) + 1 = 2$$

This is correct! The 3rd term of the Fibonacci series is 2.

For input value 4

If the user enters a value of 4, the following recursive calls are made.



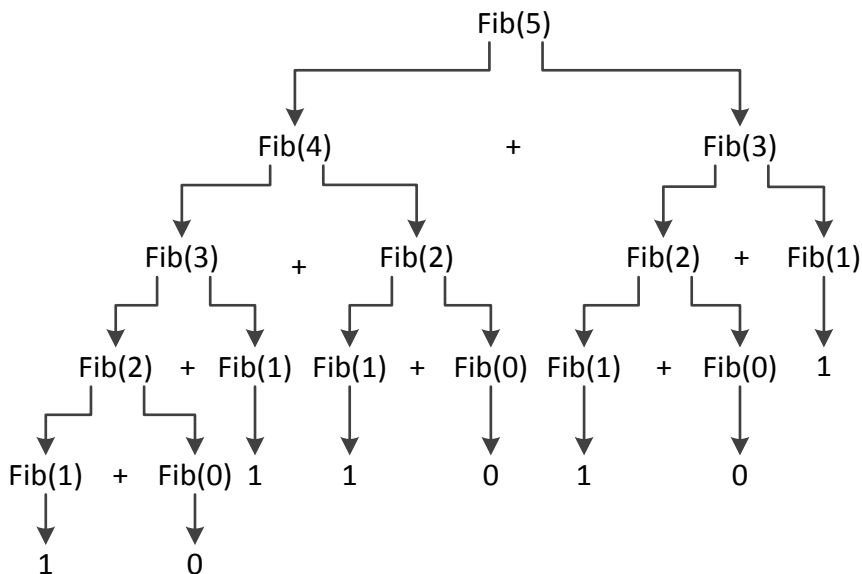
If you calculate the results of these recursive calls, you have

$$((1 + 0) + 1) + (1 + 0) = 3$$

This is also correct! The 4th term of the Fibonacci series is 3.

For input value 5

If the user enters a value of 5, the following recursive calls are made.



If you calculate the results of these recursive calls, you have

$$(((1 + 0) + 1) + (1 + 0)) + ((1 + 0) + 1) = 5$$

This is correct as well! The 5th term of the Fibonacci series is 5.

Notice: Please note that for each succeeding term of the Fibonacci series, the number of calls increments exponentially. For example, the calculation of the 40th or the 50th term of the Fibonacci series using this function can prove tragically slow!

Recursion helps you write more creative and more elegant scripts, but keep in mind that it is not always the best option. The main disadvantage of recursion is that it is hard for a programmer to think through the logic, and therefore it is difficult to debug a code that contains a recursive subprogram. Furthermore, a recursive algorithm may prove worse than a non-recursive algorithm because it may consume too much CPU time or too much main memory (RAM). So, there are times where it would be better to follow the KISS principle and, instead of using a recursion, solve the algorithm using loop control structures.

Notice: For you who don't know what the KISS principle is, it is an acronym for "Keep It Simple, Stupid!" It states that most systems work best if they are kept simple, avoiding any unnecessary complexity!

41.9 Review Questions: True/False

Choose **true** or **false** for each of the following statements.

1. Each subprogram uses its own memory space to hold the values of its variables.
2. Variables used in a subprogram "live" as long as the subprogram is being executed.
3. The only variables that "live" for as long as the PHP script is being executed are the variables of the main code.
4. A subprogram can call the main code.
5. If an argument is passed by value and its value is changed within the subprogram, it does not get changed outside of it.
6. The name of an actual argument and the name of the corresponding formal argument must be the same.
7. The total number of formal and actual arguments must be the same.
8. An expression can be passed by reference.
9. Arrays in PHP can be passed by reference.
10. You can pass an array to a procedure but a procedure cannot return an array to the caller.
11. A function can accept an array through its formal argument list.
12. A procedure can call any function.
13. A function can call any procedure.
14. Within a statement, a function can be called only once.
15. A procedure can return a value through its formal argument list.
16. A subprogram can be called by another subprogram or by the main code.

17. If you assign a default value to an argument within the formal argument list, it means that no matter what value is passed for that argument, the default value is used.
18. An argument is called an optional argument when a default value is assigned to that argument within the actual argument list.
19. Optional arguments should be on the left side of any non-optional arguments.
20. The default value of an argument can be an expression.
21. The scope of a variable refers to the range of effect of that variable.
22. If the value of a global variable is altered within a subprogram, this change is reflected outside the subprogram as well.
23. You can have two variables of global scope of the same name.
24. Recursion is a programming technique in which a subprogram calls itself.
25. A recursive algorithm must have a base case.
26. Using recursion to solve a specific problem is not always the best option.

41.10 Review Exercises

Complete the following exercises.

1. Without using a trace table, can you find out what the next PHP script displays?

```
<?php
function f1() {
    $a = 22;
}
function f2() {
    $a = 33;
}

$a = 5;
f1();
f2();
echo $a;
?>
```

2. Without using a trace table, can you find out what the next PHP script displays?

```
<?php
function f1($number1) {
    return 2 * $number1;
}
```

```
function f2($number1, $number2) {  
    return f1($number1) + f1($number2);  
}  
  
$a = 3;  
$b = 4;  
  
echo f2($a, $b);  
?>
```

3. Without using a trace table, can you find out what the next PHP script displays?

```
<?php  
function f1(&$number1) {  
    $number1 *= 2;  
}  
  
function f2($number1, $number2) {  
    f1($number1);  
    f1($number2);  
    return $number1 + $number2;  
}  
  
$a = 2;  
$b = 5;  
  
echo f2($a, $b);  
?>
```

4. Create a trace table to determine the values of the variables in each step of the following PHP script when the value 12 is entered.

```
<?php  
function swap(&$x, &$y) {  
    $temp = $x;  
    $x = $y;  
    $y = $temp;  
}  
  
$k = trim(fgets(STDIN));  
$m = 1;  
$a = 1;  
while ($a < 8) {
```

```
    if ($k % $m != 0) {
        $x = $a % $m;
        swap ($m, $a);
    }
    else {
        $x = $a + $m + intval($a - $m);
    }

    echo $m, " ", $a, " ", $x, "\n";

    $a += 2;
    $m++;
    swap ($a, $m);
}
?>
```

5. Without using a trace table, can you find out what the next PHP script displays?

```
<?php
function display($str = "hello") {
    $str = str_replace("a", "e", $str);
    echo $str;
}

display("hello");
display();
display("hallo");
?>
```

6. Without using a trace table, can you find out what the next PHP script displays?

```
<?php
function f1() {
    global $a, $b;
    $a = $a + $b;
}

$a = 10;
$b = 5;
f1();
$b--;
echo $a;
```

```
?>
```

7. Without using a trace table, can you find out what the next PHP script displays?

```
<?php
function f1() {
    global $a, $b;
    $a = $a + $b;
    f2();
}

function f2() {
    global $a, $b;
    $a = $a + $b;
    f3();
}

function f3() {
    global $a, $b;
    $a = $a + $b;
}

$a = 3;
$b = 4;
f1();
echo $a, " ", $b;
?>
```

8. For the following PHP script, convert the parts marked with a dashed rectangle into subprograms.

```
<?php
define("STUDENTS", 10);
define("LESSONS", 5);

$names = array();
$grades = array(array());
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo "Enter name for student No. ", $i + 1, ": ";
    $names[$i] = trim(fgets(STDIN));

    for ($j = 0; $j <= LESSONS - 1; $j++) {
        echo "Enter grade for lesson No. ", $j + 1, ": ";
    }
}
```

```

    $grades[$i][$j] = trim(fgets(STDIN));
}
}

```

```

$average = array();
for ($i = 0; $i <= STUDENTS - 1; $i++) {
    $average[$i] = 0;
    for ($j = 0; $j <= LESSONS - 1; $j++) {
        $average[$i] += $grades[$i][$j];
    }
    $average[$i] /= LESSONS;
}

```

```

for ($m = 1; $m <= STUDENTS - 1; $m++) {
    for ($n = STUDENTS - 1; $n >= $m; $n--) {
        if ($average[$n] > $average[$n - 1]) {
            $temp = $average[$n];
            $average[$n] = $average[$n - 1];
            $average[$n - 1] = $temp;

            $temp = $names[$n];
            $names[$n] = $names[$n - 1];
            $names[$n - 1] = $temp;
        }
        elseif ($average[$n] == $average[$n - 1]) {
            if ($names[$n] < $names[$n - 1]) {
                $temp = $names[$n];
                $names[$n] = $names[$n - 1];
                $names[$n - 1] = $temp;
            }
        }
    }
}
}

```

```

for ($i = 0; $i <= STUDENTS - 1; $i++) {
    echo $names[$i], "\t", $average[$i], "\n";
}

```

```

?>

```

9. For the following PHP script, convert the parts marked with a dashed rectangle into subprograms.

```
<?php
```

```
echo "Enter a message: ";
$message = trim(fgets(STDIN));
$message = strtolower($message);
```

```
$last_pos = strlen($message) - 1;
```

```
$message_clean = "";
for ($i = 0; $i <= $last_pos; $i++) {
    if ($message[$i] != " " && $message[$i] != "," &&
        $message[$i] != "." && $message[$i] != "?") {

        $message_clean .= $message[$i];
    }
}
```

```
$middle_pos = intval((strlen($message_clean) - 1) / 2);
$j = strlen($message_clean) - 1;
$palindrome = true;
for ($i = 0; $i <= $middle_pos; $i++) {
    if ($message_clean[$i] != $message_clean[$j]) {
        $palindrome = false;
        break;
    }
    $j--;
}
```

```
if ($palindrome == true) {
    echo "The message is palindrome";
}
?>
```

10. The next PHP script finds the greatest value among four values given. Rewrite the script without using subprograms.

```
<?php
```

```
function my_max($n, &$m) {
    if ($n > $m) {
        $m = $n;
    }
}
```

```
}

$a = trim(fgets(STDIN));
$b = trim(fgets(STDIN));
$c = trim(fgets(STDIN));
$d = trim(fgets(STDIN));

$max = $a;
my_max($b, $max);
my_max($c, $max);
my_max($d, $max);

echo $max;

?>
```

11. Write a subprogram that accepts three numbers through its formal argument list and then returns their sum and their average.
12. Write a subprogram named `my_round()` that accepts a real and an integer through its formal argument list and then returns the real rounded to as many decimal places as the integer indicates. Moreover, if no value is passed for the integer, the subprogram should return the real rounded to two decimal places by default. Try not to use the `round()` function of PHP.
13. Do the following:
 - i. Write a subprogram named `get_input()` that prompts the user to enter an answer “yes” or “no” and then returns the value `true` or `false` correspondingly to the caller. Make the subprogram accept the answer in all possible forms such as “yes”, “YES”, “Yes”, “No”, “NO”, “no”, and so on.
 - ii. Write a subprogram named `find_area()` that accepts the base and the height of a parallelogram through its formal argument list and then returns its area.
 - iii. Using the subprograms `get_input()` and `find_area()` cited above, write a PHP script that prompts the user to enter the base and the height of a parallelogram and then calculates and displays its area. The script should iterate as many times as the user wishes. At the end of each calculation, the script should ask the user whether he or she wishes to calculate the area of another parallelogram. If the answer is “yes” the script should repeat.
14. Do the following:
 - i. Write a subprogram named `get_arrays()` that prompts the user to enter the grades and the names of 100 students into the arrays `$grades` and `$names`, correspondingly. The two arrays must be returned to the caller.

- ii. Write a subprogram `average()` that accepts the array `$grades` through its formal argument list and returns the average grade.
 - iii. Write a subprogram named `sort_arrays()` that accepts the arrays `$grades` and `$names` through its formal argument list and sorts the array `$grades` in descending order using the insertion sort algorithm. The subprogram must preserve the relationship between the elements of the two arrays.
 - iv. Using the three subprograms cited above, write a PHP script that prompts the user to enter the grades and the names of 100 students and then displays all student names whose grade is less than the average grade, sorted by grade in descending order.
15. In a song contest, there is an artist who is scored by 10 judges. However, according to the rules of this contest, the total score is calculated after excluding the maximum and the minimum score. Do the following:
 - i. Write a subprogram named `get_array()` that prompts the user to enter the scores of the 10 judges into an array and then returns the array to the caller.
 - ii. Write a subprogram named `find_min_max()` that accepts an array through its formal argument list and then returns the maximum and the minimum value.
 - iii. Using the subprograms `get_array()` and `find_min_max()` cited above, write a PHP script that lets the user enter the name of the artist and the score he or she gets from each judge. The script should then display the message “Artist NN got XX points” where NN and XX must be replaced by actual values.
16. On a chessboard you must place grains of wheat on each square, such that one grain is placed on the first square, two on the second, four on the third, and so on (doubling the number of grains on each subsequent square). Do the following:
 - i. Write a recursive function named `woc()` that accepts the index of a square and returns the number of grains of wheat that are on this square. Since a chessboard contains $8 \times 8 = 64$ squares, assume that the index is an integer between 1 and 64.
 - ii. Using the function `woc()` cited above, write a PHP script that calculates and displays the total number of grains of wheat that are on the chessboard in the end.
17. Do the following:
 - i. Write a recursive function named `factorial()` that accepts an integer through its formal argument list and then returns its factorial.
 - ii. Using the function `factorial()` cited above, write a recursive function named `my_cos()` that calculates and returns the cosine of x using the Taylor series, shown next.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + \frac{x^{40}}{40!}$$

Hint: Keep in mind that x is in radians and $\frac{x^0}{0!} = 1$.

- iii. Using the function `my_cos()` cited above, write a PHP script that calculates and displays the cosine of 45° .

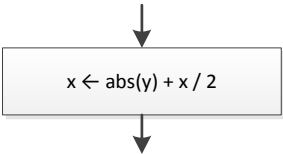
Chapter 42

Flowcharts with Subprograms

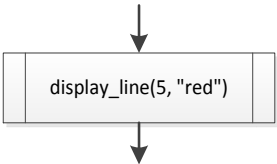
42.1 Designing and Calling Sub-Algorithms in Flowcharts

First of all, let's clarify something. A sub-algorithm in a flowchart is what a subprogram is in a program—that is, a block of statements packaged as a unit that performs a specific task. Thus, a sub-algorithm can be a function or a procedure. In this chapter, you will learn how a sub-algorithm and a call to a sub-algorithm can be represented in flowcharts.

As you already know, a call to a function—whether built-in or user-defined—is made in the same way. You enter the function's name followed by a list of arguments (if necessary), either within a statement that assigns function's returned value to a variable or directly within an expression. Of course, the same applies to calling a function when a flowchart is used. The next example calls the built-in function `abs()`.



As in PHP, a call to a procedure is different from a call to a function. You call a procedure by just entering its name within a predefined process symbol. Does this symbol ring a bell to you? Of course it does. This is one of those symbols that you learned about in Chapter 4 but that you haven't used until now. The next example calls a user-defined procedure named `display_line()` and passes two arguments.

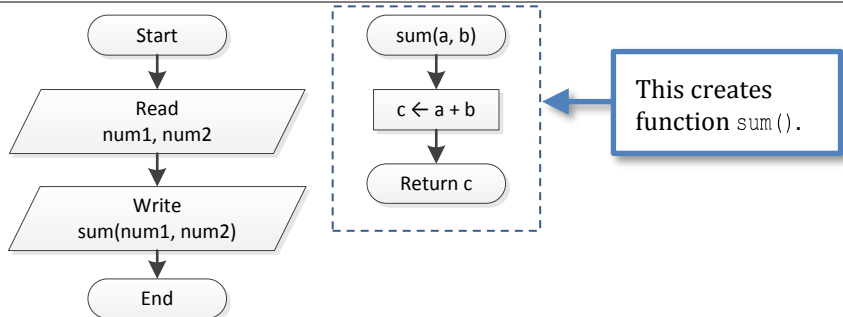


Remember! The predefined process symbol depicts procedures that are formally defined elsewhere, such as in a separate flowchart. The predefined process symbol has one entrance and one exit.

But how do you represent a sub-algorithm in a flowchart? Below, you can find some examples.

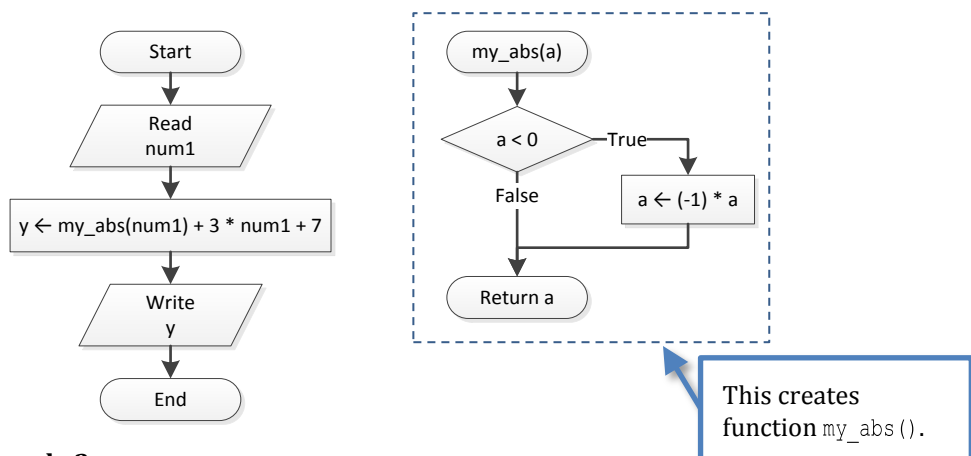
Example 1

In this example the main algorithm calls the function `sum()`, which calculates and returns the sum of two numbers.



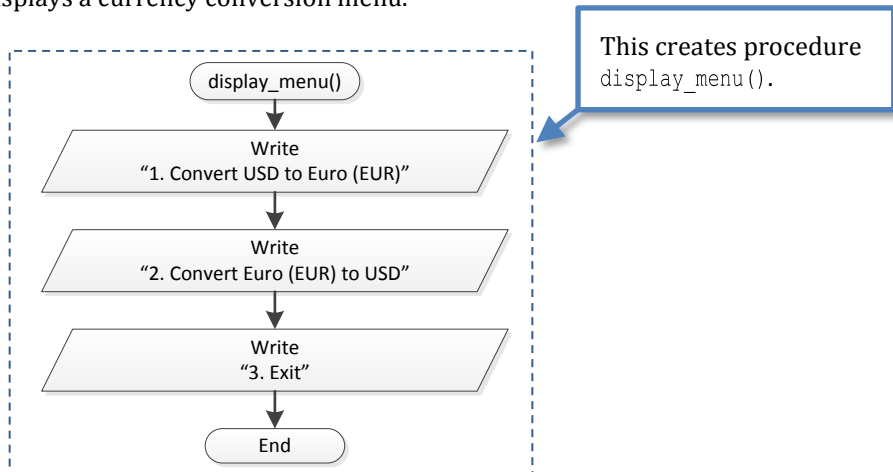
Example 2

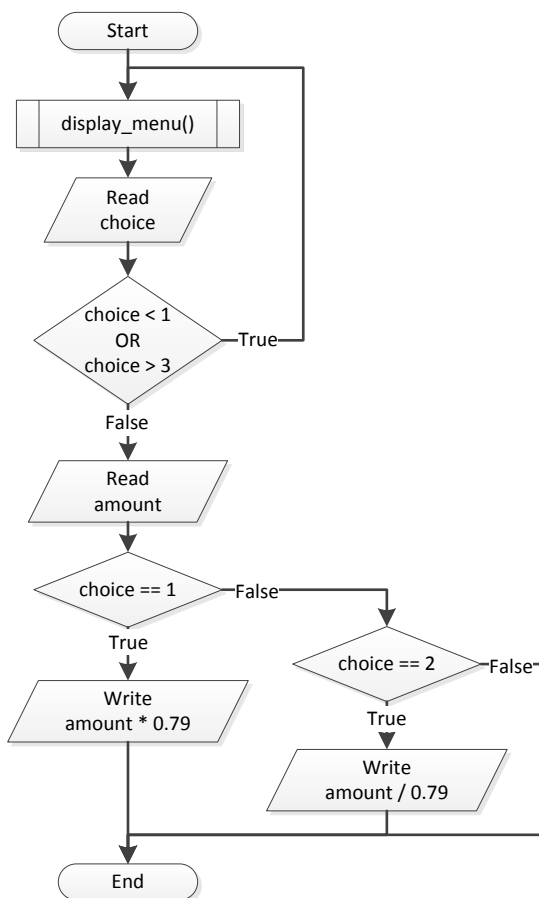
In this example the main algorithm calls the function `my_abs()`, which calculates and returns the absolute value of a number.



Example 3

In this example the main algorithm calls the procedure `display_menu()`, which displays a currency conversion menu.





42.2 Converting PHP Scripts to Flowcharts

Now, with everything clarified, let's directly start solving some exercises.

Exercise 42.2-1 *Designing the Flowchart*

Design the flowchart that corresponds to the following PHP script.

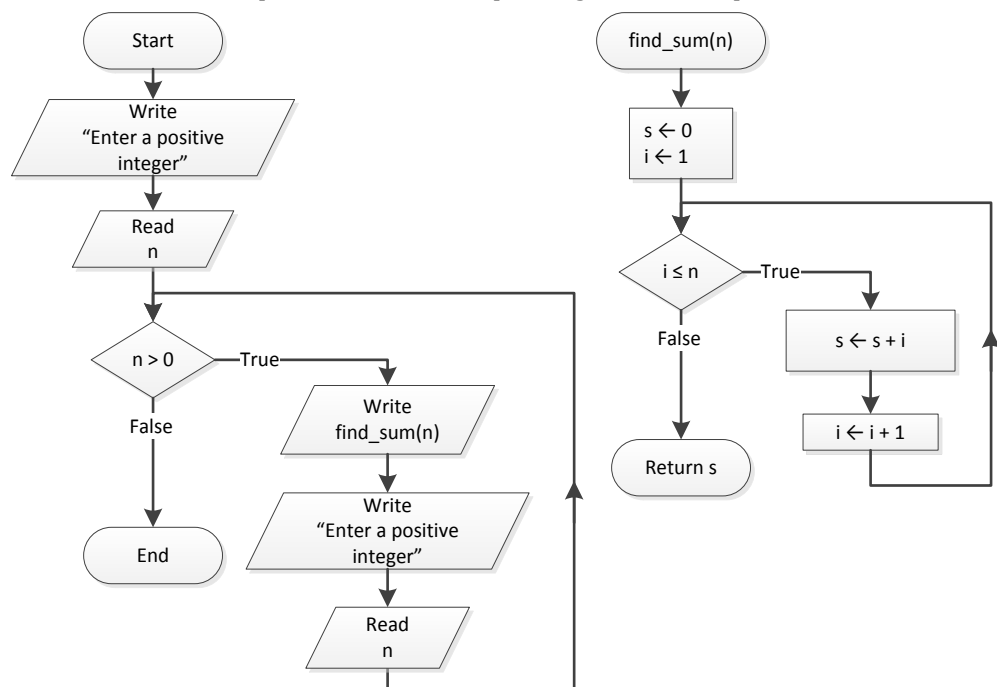
```

<?php
function find_sum($n) {
    $s = 0;
    for ($i = 1; $i <= $n; $i++) {
        $s = $s + $i;
    }
    return $s;
}

echo "Enter a positive integer";
  
```

```
$n = trim(fgets(STDIN));
while ($n > 0) {
    echo find_sum($n);
    echo "Enter a positive integer";
    $n = trim(fgets(STDIN));
}
?>
```

Without further explanation, the corresponding flowchart is presented here.



Exercise 42.2-2 Designing the Flowchart

Design the flowchart that corresponds to the following code fragment.

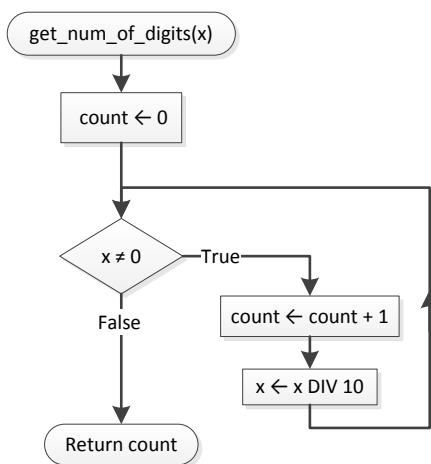
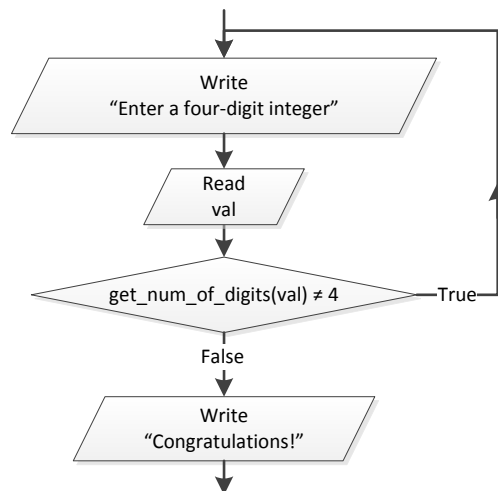
```
function get_num_of_digits($x) {
    $count = 0;
    while ($x != 0) {
        $count++;
        $x = intval($x / 10);
    }
    return $count;
}
```

```
do {
    echo "Enter a four-digit integer";
    $val = trim(fgets(STDIN));
} while (get_num_of_digits($val) != 4);

echo "Congratulations!";
```

Solution

The solution is presented next



Exercise 42.2-3 Designing the Flowchart

Design the flowchart that corresponds to the following code fragment.

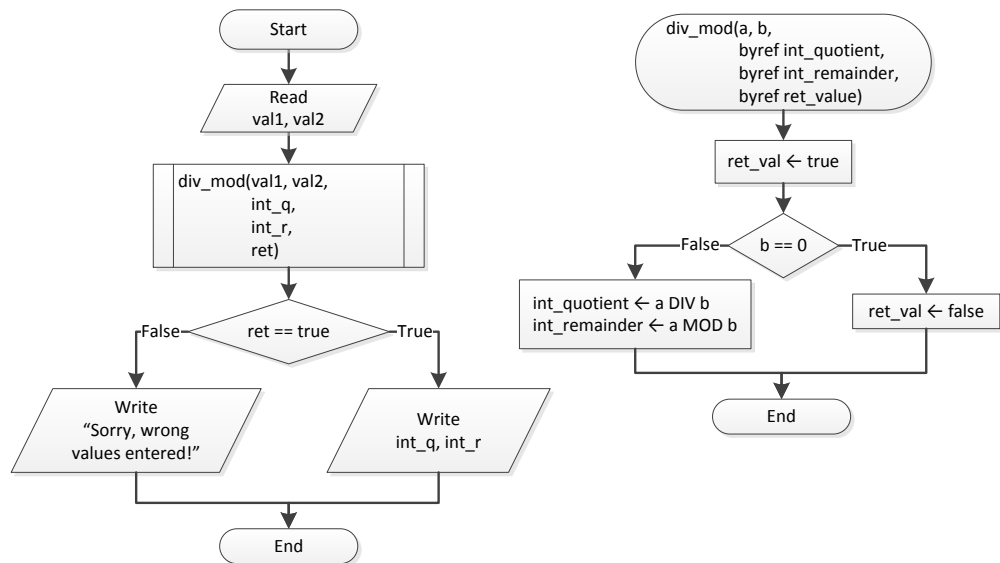
```
<?php
function div_mod($a, $b, &$int_quotient, &$int_remainder, &$ret_value) {
    $ret_value = true;
    if ($b == 0) {
        $ret_value = false;
    }
    else {
        $int_quotient = intval($a / $b);
        $int_remainder = $a % $b;
    }
}
```

```
$val1 = trim(fgets(STDIN));
$val2 = trim(fgets(STDIN));
div_mod($val1, $val2, $int_q, $int_r, $ret);
if ($ret == true) {
    echo $int_q, $int_r;
}
else {
    echo "Sorry, wrong values entered!";
}
?>
```

Solution

In this PHP script, the procedure `div_mod()` indirectly returns three values through its arguments `$int_quotient`, `$int_remainder`, and `$ret_value`. This occurs because values to those arguments are passed by reference.

As stated many times in this book, flowcharts are a loose method of representing an algorithm. Thus, you can represent a pass by reference using the ampersand (&) symbol (as PHP does); however, if you do so, you take the risk that someone reading your flowchart might not understand what the ampersand (&) symbol does. So, it is better to write the keyword `byref` instead, which is more obvious and more clearly denotes what it actually does.

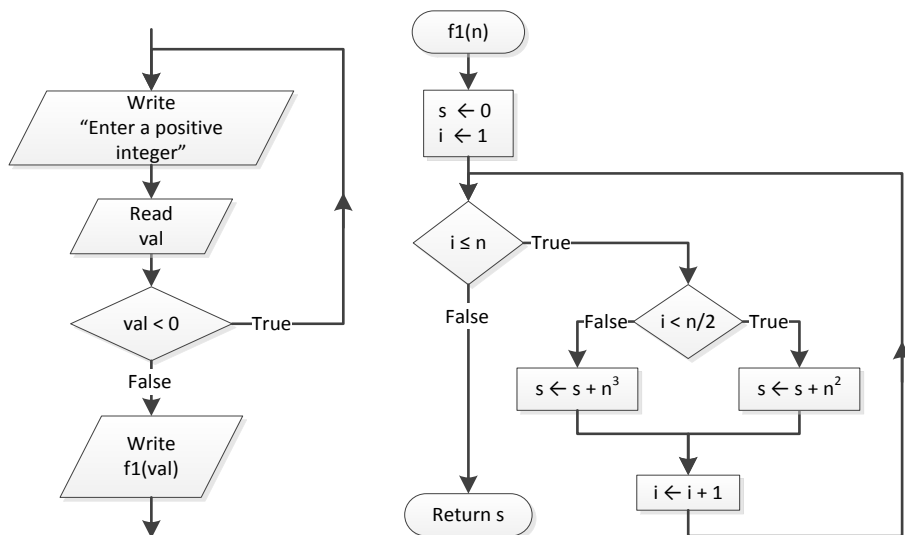


Notice: Some programmers, instead of using the keyword `byref`, prefer to write the keyword `inout`, which denotes pretty much the same thing— that the variable is both input (it accepts values) and output (it returns values).

42.3 Converting Flowcharts to PHP Scripts

Exercise 42.3-1 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart fragment.



Solution

This exercise is quite simple. Without further explanation, the code fragment is shown here.

```

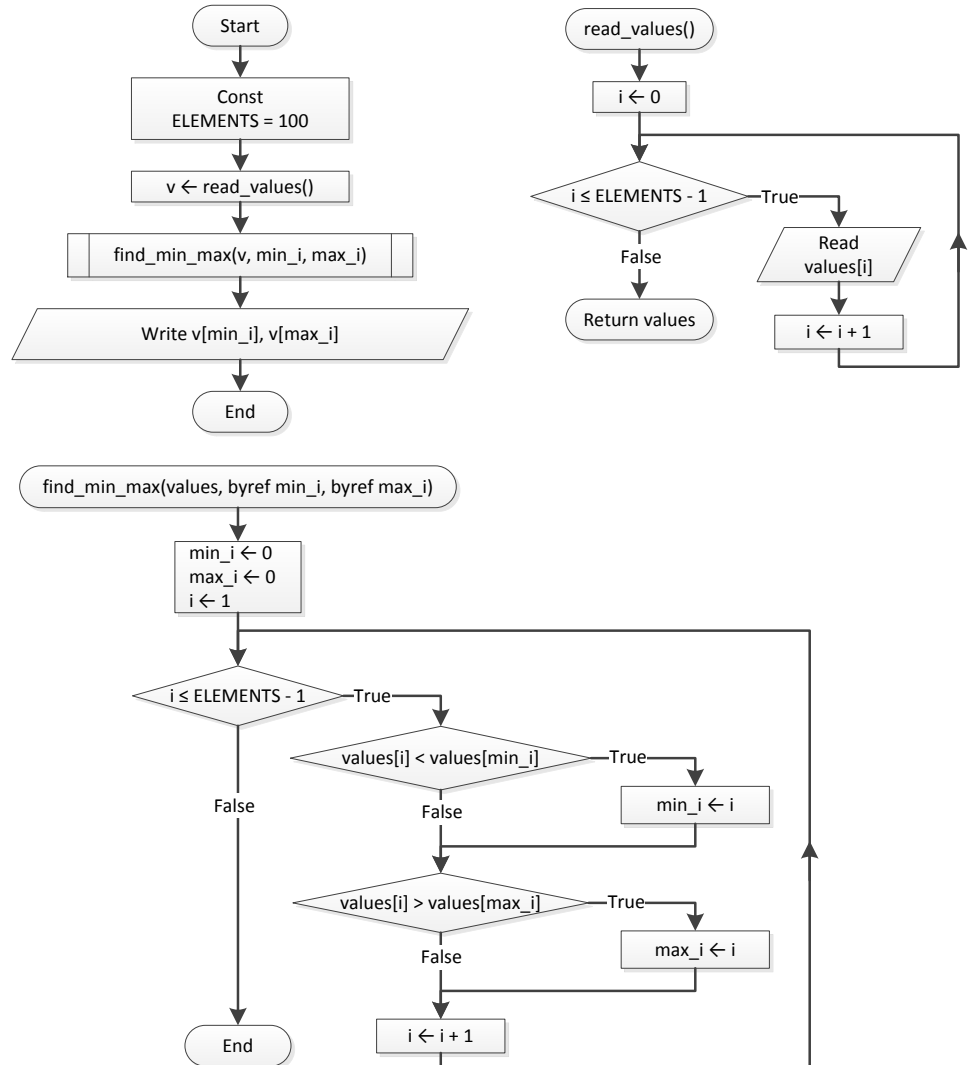
function f1($n) {
    $s = 0;
    for ($i = 1; $i <= $n; $i++) {
        if ($i < $n / 2) {
            $s += pow($n, 2);
        }
        else {
            $s += pow($n, 3);
        }
    }
    return $s;
}

```

```
do {
    echo "Enter a positive integer";
    $val = trim(fgets(STDIN));
} while ($val < 0);
echo f1($val);
```

Exercise 42.3-2 Writing the PHP Script

Write the PHP script that corresponds to the following flowchart fragment.



Solution

In this flowchart there is a function named `read_values()` and a procedure named `find_min_max()`. As you can see, the latter returns the values of variables `$max_i` and `$min_i` indirectly (they are passed by reference). The PHP script is as follows.

```
<?php
define("ELEMENTS", 100);

function read_values() {
    for ($i = 0; $i <= ELEMENTS - 1; $i++) {
        $values[$i] = trim(fgets(STDIN));
    }
    return $values;
}

function find_min_max($values, &$amp;min_i, &$amp;max_i) {
    $min_i = 0;
    $max_i = 0;
    for ($i = 1; $i <= ELEMENTS - 1; $i++) {
        if ($values[$i] < $values[$min_i]) {
            $min_i = $i;
        }
        if ($values[$i] > $values[$max_i]) {
            $max_i = $i;
        }
    }
}

$v = read_values();
find_min_max($v, $min_i, $max_i);
echo $v[$min_i], $v[$max_i];
?>
```

Notice: Please study the way the procedure `find_min_max()` finds the index positions of the minimum and the maximum values of the array `$values`. This method is not the same as the one you learned in Chapter 37; however, it can be used as an alternative.

42.4 Review Exercises

Complete the following exercises.

1. Design the flowchart that corresponds to the following PHP script.

```
<?php
function test_integer($number) {
    $return_value = false;
    if ($number == intval($number)) {
        $return_value = true;
    }
    return $return_value;
}

function test_positive($number) {
    $return_value = false;
    if ($number > 0) {
        $return_value = true;
    }
    return $return_value;
}

$sum = 0;
$count = 0;
$x = trim(fgets(STDIN));
while(test_positive($x) == true) {
    if (test_integer($x) == true) {
        $sum += $x;
        $count++;
    }
    $x = trim(fgets(STDIN));
}

if ($count > 0) {
    echo $sum / $count;
}

?>
```

2. Design the flowchart that corresponds to the following PHP script.

```
<?php
define("PEOPLE", 30);
```

```
function get_age() {
    do {
        $x = trim(fgets(STDIN));
    } while (is_numeric($x) != true || $x <= 0);
    return $x;
}

function find_max($a) {
    $max_i = 0;
    for ($i = 1; $i <= PEOPLE - 1; $i++) {
        if ($a[$i] > $a[$max_i]) {
            $max_i = $i;
        }
    }
    return $max_i;
}

$first_names = array();
$last_names = array();
$ages = array();
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    $first_names[$i] = trim(fgets(STDIN));
    $last_names[$i] = trim(fgets(STDIN));
    $ages[$i] = get_age();
}

$index_of_max = find_max($ages);

echo $first_names[$index_of_max];
echo $last_names[$index_of_max];
echo $ages[$index_of_max];
?>
```

3. Design the flowchart that corresponds to the following PHP script.

```
<?php
define("PEOPLE", 40);

function my_swap(&$x, &$y) {
    $temp = $x;
    $x = $y;
    $y = $temp;
```

```

    }

    function my_sort(&$a) {
        for ($m = 1; $m <= PEOPLE - 1; $m++) {
            for ($n = PEOPLE - 1; $n >= $m; $n--) {
                if ($a[$n] < $a[$n - 1]) {
                    my_swap($a[$n], $a[$n-1]);
                }
            }
        }
    }

    function display_array($a, $ascending) {
        if ($ascending == true) {
            for ($i = 0; $i <= PEOPLE - 1; $i++) {
                echo $a[$i], "\n";
            }
        }
        else {
            for ($i = PEOPLE - 1; $i >= 0; $i--) {
                echo $a[$i], "\n";
            }
        }
    }

    $names = array();
    for ($i = 0; $i <= PEOPLE - 1; $i++) {
        $names[$i] = trim(fgets(STDIN));
    }

    my_sort($names);
    display_array($names, true);
    display_array($names, false);
}
?>

```

4. Design the flowchart that corresponds to the following PHP script.

```

<?php
function get_consumption() {
    echo "Enter kWh consumed: ";
    $kwh = trim(fgets(STDIN));
    while (is_numeric($kwh) != true || $kwh < 0) {

```

```
    echo "Error! Enter kWh consumed:  \n";
    $kwh = trim(fgets(STDIN));
}
return $kwh;
}

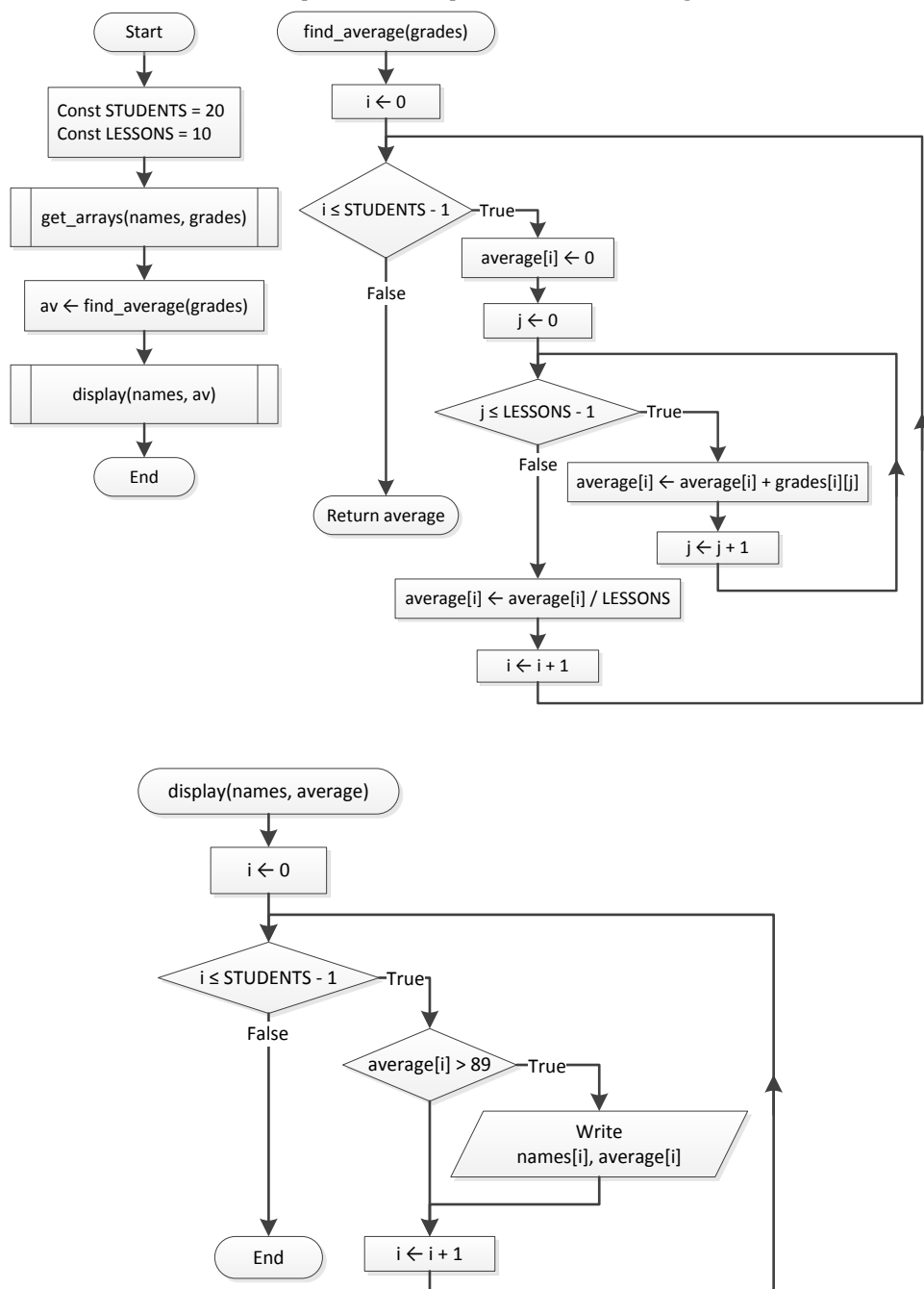
function find_amount($kwh) {
    if ($kwh <= 450) {
        $amount = $kwh * 0.07;
    }
    elseif ($kwh <= 2200) {
        $amount = 450 * 0.07 + ($kwh - 450) * 0.20;
    }
    else {
        $amount = 450 * 0.07 + 1750 * 0.20 + ($kwh - 1750) * 0.33;
    }

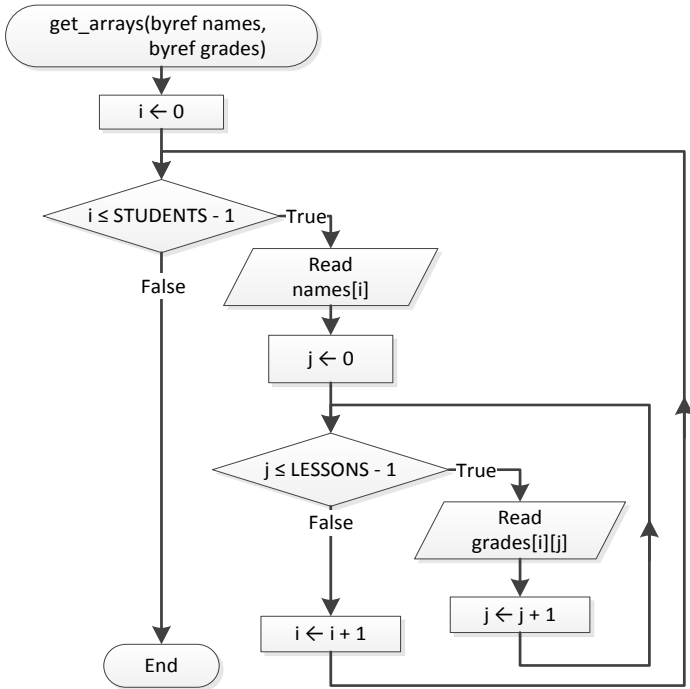
    $amount += 0.22 * $amount;
    return $amount;
}

do {
    $kwh = get_consumption();
    echo find_amount($kwh);

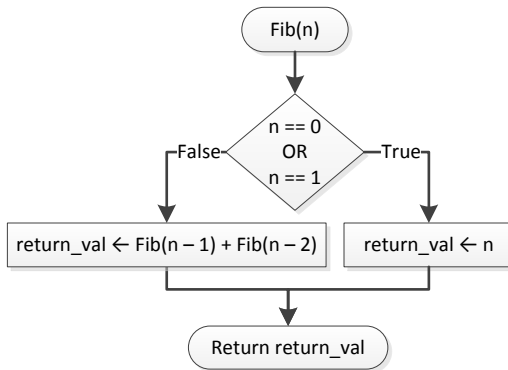
    echo "Repeat?\n";
    $answer = trim(fgets(STDIN));
} while (strtoupper($answer) == "YES");
?>
```

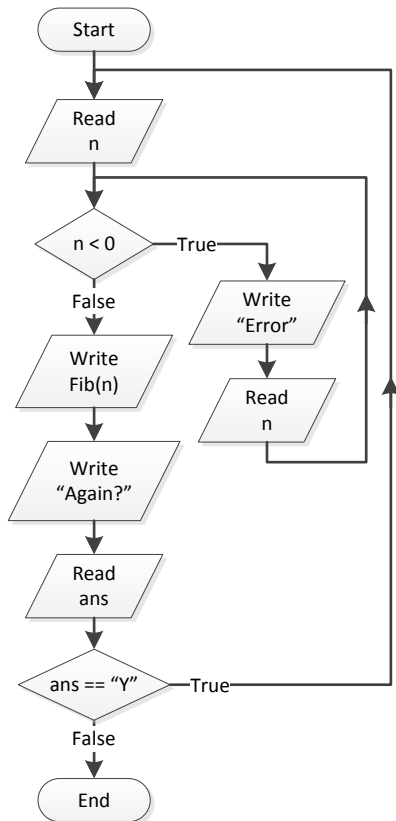
5. Write the PHP script that corresponds to the following flowchart.





6. Write the PHP script that corresponds to the following flowchart.





Chapter 43

More Exercises with Subprograms

43.1 Simple Exercises with Subprograms

Exercise 43.1-1 *Finding the Average Values of Positive Integers*

Do the following:

- i. Write a subprogram named `test_integer()` that accepts a number through its formal argument list and returns `true` when the passed number is an integer; it must return `false` otherwise.
- ii. Using the subprogram `test_integer()` cited above, write a PHP script that lets the user enter integer values repeatedly until a real one is entered. In the end, the script should display the average value of positive integers entered.

Solution

Since the subprogram `test_integer()` returns one value (`true` or `false`), it can be written as a function. The solution is presented next.

file_43.1-1.php

```
<?php
function test_integer($number) {
    $return_value = false;
    if ($number == intval($number)) {
        $return_value = true;
    }
    return $return_value;
}

$sum = 0;
$count = 0;
$x = trim(fgets(STDIN));
while(test_integer($x) == true) {
    if ($x > 0) {
        $sum += $x;
        $count++;
    }
    $x = trim(fgets(STDIN));
}

if ($count > 0) {
    echo $sum / $count;
}

?>
```

Notice: Please note the last single-alternative decision structure, if ($\$count > 0$). It is necessary in order for the script to satisfy the property of definiteness. Think about it! If the user enters a real right from the beginning, the variable $\$count$, in the end, will contain a value of zero.

Exercise 43.1-2 Finding the Sum of Odd Positive Integers

Do the following:

- i. Write a subprogram named `test_integer` that accepts a number through its formal argument list and returns `true` when the passed number is an integer; it must return `false` otherwise.
- ii. Write a subprogram named `test_odd` that accepts a number through its formal argument list and returns `true` when the passed number is odd; it must return `false` otherwise.
- iii. Write a subprogram named `test_positive` that accepts a number through its formal argument list and returns `true` when the passed number is positive; it must return `false` otherwise.
- iv. Using the three subprograms cited above, write a PHP script that lets the user enter numeric values repeatedly until a negative one is entered. In the end, the script should display the sum of odd positive integers entered.

Solution

This exercise is pretty much the same as the previous one. Each subprogram returns one value (which can be `true` or `false`) thus, all subprograms can be written as functions. The solution is presented here.

file_43.1-2.php

```
<?php
function test_integer($number) {
    $return_value = false;
    if ($number == intval($number)) {
        $return_value = true;
    }
    return $return_value;
}

function test_odd($number) {
    $return_value = false;
    if ($number % 2 != 0) {
        $return_value = true;
    }
    return $return_value;
}
```

```

}

function test_positive($number) {
    $return_value = false;
    if ($number > 0) {
        $return_value = true;
    }
    return $return_value;
}

$sum = 0;
$x = trim(fgets(STDIN));
while(test_positive($x) == true) {
    if (test_integer($x) == true && test_odd($x) == true) {
        $sum += $x;
    }
    $x = trim(fgets(STDIN));
}

echo $sum;

?>

```

Exercise 43.1-3 *Finding the Values of y*

Write a PHP script that finds and displays the values of y (if possible) in the following formula.

$$y = \begin{cases} \frac{3x}{x-5} + \frac{7-x}{2x}, & x \geq 1 \\ \frac{45-x}{x+2} + 3x, & x < 1 \end{cases}$$

For each part of the formula, write a subprogram that accepts x through its formal argument list and then calculates and displays the result. An error message should be displayed when the calculation is not possible.

Solution

Since there are two parts of the formula, two subprograms must be written. Each subprogram should calculate and display the result of the corresponding formula or display an error message when the calculation is not possible. As these two subprograms return no result, they can both be written as procedures. The solution is as follows.

file_43.1-3.php

```
<?php
function formula1($x) {
    if ($x == 5) {
        echo "Error! Division by zero";
    }
    else {
        $y = 3 * $x / ($x - 5) + (7 - $x) / (2 * $x);
        echo $y;
    }
}

function formula2($x) {
    if ($x == -2) {
        echo "Error! Division by zero";
    }
    else {
        $y = (45 - $x) / ($x + 2) + 3 * $x;
        echo $y;
    }
}

echo "Enter a value for x: ";
$x = trim(fgets(STDIN));
if ($x >= 1) {
    formula1($x);
}
else {
    formula2($x);
}
?>
```

Exercise 43.1-4 Roll, Roll, Roll the... Dice!

Do the following:

- i. Write a subprogram named `dice()` that returns a random integer between 1 and 6.
- ii. Write a subprogram named `search_and_count()` that accepts an integer and an array through its formal argument list and returns the number of times the integer exists in the array.

- iii. Using the subprograms `dice()` and `search_and_count()` cited above, write a PHP script that fills an array with 100 random integers (between 1 and 6) and then lets the user enter an integer. The script should find and display how many times that given integer exists in the array.

Solution

Both subprograms can be written as functions because they both return one value each. Function `dice()` returns a random integer between 1 and 6, and function `search_and_count()` returns a number that indicates the number of times an integer exists in an array. The solution is presented here.

file_43.1-4.php

```
<?php
define("ELEMENTS", 100);

function dice() {
    return rand(1, 6);
}

function search_and_count($x, $a) {
    $count = 0;
    for ($i = 0; $i <= ELEMENTS - 1; $i++) {
        if ($a[$i] == $x) {
            $count++;
        }
    }
    return $count;
}

$a = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $a[$i] = dice();
}

$x = trim(fgets(STDIN));
echo "Given value exists in the array ", search_and_count($x, $a), " times";
?>
```

Exercise 43.1-5 *How Many Times Does Each Number of the Dice Appear?*

Using the functions `dice()` and `search_and_count()` cited in the previous exercise, write a PHP script that fills an array with 100 random integers (between 1 and 6) and then

displays how many times each of the six numbers appears in the array, as well as which number appears most often.

Solution

If you were to solve this exercise without using loop control structures, it would be something like the following.

```
//Variable $n1 is assigned the number of times that
//value 1 exists in array $a
$n1 = search_and_count(1, $a);
//Variable $n2 is assigned the number of times that
//value 2 exists in array $a
$n2 = search_and_count(2, $a);
.
.
.
//Variable $n6 is assigned the number of times that
//value 6 exists in array $a
$n6 = search_and_count(6, $a);

//Display how many times each of the six numbers appears in array $a
echo $n1, $n2, $n3, $n4, $n5, $n6;

//Find maximum of $n1, $n2,... $n6
$max = $n1;
$max_i = 1;

if ($n2 > $max) {
    $max = $n2;
    $max_i = 2;
}
if ($n3 > $max) {
    $max = $n3;
    $max_i = 2;
}
.
.
.
if ($n6 > $max) {
    $max = $n6;
    $max_i = 6;
```

```
}

//Display which number appears in the array most often.
echo $max_i;
```

But now that you are reaching the end of the book, of course, you can do something more creative. Instead of assigning each result of the `search_and_count()` function to individual variables `$n1`, `$n2`, `$n3`, `$n4`, `$n5`, and `$n6`, you can assign those results to the positions 0, 1, 2, 3, 4, and 5 of an array named `$n`, as shown here.

```
$n = array();
for ($i = 0; $i <= 5; $i++) {
    $n[$i] = search_and_count($i + 1, $a);
}
```

Next, of course, you can find the maximum of the array `$n` as usual. The complete solution is shown here.

file_43.1-5.php

```
<?php
define("ELEMENTS", 100);

function dice() {
    return rand(1, 6);
}

function search_and_count($x, $a) {
    $count = 0;
    for ($i = 0; $i <= ELEMENTS - 1; $i++) {
        if ($a[$i] == $x) {
            $count++;
        }
    }
    return $count;
}

//Create array $a of random integers between 1 and 6
$a = array();
for ($i = 0; $i <= ELEMENTS - 1; $i++) {
    $a[] = dice();
}

//Create array $n and
```

```
//display how many times each of the six numbers appears in array $a
$n = array();
for ($i = 0; $i <= 5; $i++) {
    $n[$i] = search_and_count($i + 1, $a);
    echo "Value ", $i + 1, " appears ", $n[$i], " times\n";
}

//Find maximum of array $n
$max = $n[0];
$max_i = 0;
for ($i = 1; $i <= 5; $i++) {
    if ($n[$i] > $max) {
        $max = $n[$i];
        $max_i = $i;
    }
}

//Display which number appears in the array most often.
echo "Value ", $max_i + 1, " appears in the array ", $max, " times.";
?>
```

43.2 Exercises of a General Nature with Subprograms

Exercise 43.2-1 Validating Data Input

Do the following:

- i. Write a subprogram named `get_age()` that prompts the user to enter his or her age and returns it. Moreover, the subprogram should validate data input and display an error message when the user enters any non-numeric or non-positive values.
- ii. Write a subprogram named `find_max()` that accepts an array through its formal argument list and returns the index position of the maximum value of the array.
- iii. Using the subprograms `get_age()` and `find_max()` cited above, write a PHP script that prompts the user to enter the first names, last names, and ages of 50 people into an array and then finds and displays the name of the oldest person.

Solution

Since the subprogram `get_age()` returns one value, it can be written as a function. The same applies to subprogram `find_max()` because it also returns one value. The main code should prompt the user to enter the first names, the last names, and the

ages of 50 people into arrays `$first_names`, `$last_names`, and `$ages` respectively. Then, with the help of function `find_max()`, it can find the index position of the maximum value of array `$ages`. The solution is shown here.

file_43.2-1.php

```
<?php
define("PEOPLE", 50);

function get_age() {
    echo "Enter an age: ";
    $x = trim(fgets(STDIN));
    while (is_numeric($x) != true || $x <= 0) {
        echo "Error: Invalid age! \n";
        echo "Enter a positive number: ";
        $x = trim(fgets(STDIN));
    }
    return $x;
}

function find_max($a) {
    $max = $a[0];
    $max_i = 0;
    for ($i = 1; $i <= PEOPLE - 1; $i++) {
        if ($a[$i] > $max) {
            $max = $a[$i];
            $max_i = $i;
        }
    }
    return $max_i;
}

$first_names = array();
$last_names = array();
$ages = array();
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    echo "Enter first name of person No ", $i + 1, ": ";
    $first_names[$i] = trim(fgets(STDIN));
    echo "Enter last name of person No ", $i + 1, ": ";
    $last_names[$i] = trim(fgets(STDIN));
    $ages[$i] = get_age();
}
```

```
$index_of_max = find_max($ages);

echo "The oldest person is:\n";
echo $first_names[$index_of_max], " ", $last_names[$index_of_max],
" \n";
echo "He or she is ", $ages[$index_of_max], " years old!";
?>
```

Exercise 43.2-2 *Sorting an Array*

Do the following:

- i. Write a subprogram named `my_swap()` that swaps and returns the values of two arguments.
- ii. Using the subprogram `my_swap()` cited above, write a subprogram named `my_sort()` that accepts an array through its formal argument list and then sorts the array using the bubble sort algorithm. It must be able to sort in either ascending or descending order. To do this, include an additional Boolean variable within the formal argument list.
- iii. Write a subprogram `display_array()` that accepts an array through its formal argument list and then displays it.
- iv. Using the subprograms `my_sort()` and `display_array()` cited above, write a PHP script that prompts the user to enter the names of 20 people and then displays them twice: once sorted in ascending order, and once in descending order.

Solution

As you can see in the PHP script below, the procedure `my_sort()` uses an adapted version of the bubble sort algorithm. When the value `true`, or even no value, is passed to the argument `$ascending`, the algorithm sorts array `$a` in ascending order. When the value `false` is passed, the algorithm sorts array `$a` in descending order.

Moreover, the procedure `my_sort()` calls the procedure `my_swap()` every time a swap is required between the contents of two elements.

file_43.2-2.php

```
<?php
define("PEOPLE", 20);

function my_swap(&$x, &$y) {
    $temp = $x;
    $x = $y;
    $y = $temp;
```

```
}

function my_sort(&$a, $ascending = true) {
    for ($m = 1; $m <= PEOPLE - 1; $m++) {
        for ($n = PEOPLE - 1; $n >= $m; $n--) {
            if ($ascending == true) {
                if ($a[$n] < $a[$n - 1]) {
                    my_swap($a[$n], $a[$n-1]);
                }
            }
            else {
                if ($a[$n] > $a[$n - 1]) {
                    my_swap($a[$n], $a[$n-1]);
                }
            }
        }
    }
}

function display_array($a) {
    for ($i = 0; $i <= PEOPLE - 1; $i++) {
        echo $a[$i], "\n";
    }
}

$names = array();
for ($i = 0; $i <= PEOPLE - 1; $i++) {
    echo "Enter a name: ";
    $names[$i] = trim(fgets(STDIN));
}

my_sort($names);                //Sort names in ascending order
display_array($names);          //and display them

my_sort($names, false);         //Sort names in descending order
display_array($names)           //and display them.

?>
```

Notice: Please note that the argument `$ascending` is an optional argument. This means that if no value is passed for that argument, the default value `true` is used.

Exercise 43.2-3 *Progressive Rates and Electricity Consumption*

The LAV Electricity Company charges subscribers for their electricity consumption according to the following table (monthly rates for domestic accounts).

Kilowatt-hours (kWh)	USD per kWh
kWh < 400	\$0.08
401 ≤ kWh < 1500	\$0.22
1501 ≤ kWh < 3000	\$0.35
3001 ≤ kWh	\$0.50

Do the following:

- i. Write a subprogram named `get_consumption()` that prompts the user to enter the total number of kWh consumed and then returns it. Moreover, the subprogram should validate data input and display an error message when the user enters any non-numeric or negative values.
- ii. Write a subprogram named `find_amount()` that accepts kWh consumed through its formal argument list and then returns the total amount to pay.
- iii. Using the subprograms `get_consumption()` and `find_amount()` cited above, write a PHP script that prompts the user to enter the total number of kWh consumed and then calculates and displays the total amount to pay. The script should iterate as many times as the user wishes. At the end of each calculation, the script should ask the user if he or she wishes to calculate the total amount to pay for another consumer. If the answer is “yes” the script should repeat; it should end otherwise. Make your script accept the answer in all possible forms such as “yes”, “YES”, “Yes”, or even “YeS”.

Please note that the rates are progressive and that transmission services and distribution charges, as well as federal, state, and local taxes, add a total of 26% to each bill.

Solution

There is nothing new here. Processing progressive rates is something that you have already learned! If this doesn't ring any bells, you need to refresh your memory and review the corresponding exercises in Chapter 23.

The PHP script is as follows.

file_43.2-3.php

```
<?php
function get_consumption() {
    echo "Enter kWh consumed: ";
    $kwh = trim(fgets(STDIN));
    while (is_numeric($kwh) != true || $kwh < 0) {
        echo "Error: Invalid number! \n";
```

```
    echo "Enter a non-negative number: ";
    $kwh = trim(fgets(STDIN));
}
return $kwh;
}

function find_amount($kwh) {
    if ($kwh <= 400) {
        $amount = $kwh * 0.08;
    }
    elseif ($kwh <= 2000) {
        $amount = 400 * 0.08 + ($kwh - 400) * 0.22;
    }
    elseif ($kwh <= 4000) {
        $amount = 400 * 0.08 + 1100 * 0.22 + ($kwh - 1500) * 0.35;
    }
    else {
        $amount = 400 * 0.08 + 1100 * 0.22 + 1500 * 0.35 + ($kwh - 3000) * 0.50;
    }

    $amount += 0.26 * $amount;
    return $amount;
}

do {
    $kwh = get_consumption();
    echo "You need to pay: $", find_amount($kwh), "\n";

    echo "Would you like to repeat?\n";
    $answer = trim(fgets(STDIN));
} while (strtoupper($answer) == "YES");
?>
```

43.3 Review Exercises

Complete the following exercises.

1. Do the following:
 - i. Write a subprogram named `factorial()` that accepts an integer through its formal argument list and returns its factorial.
 - ii. Using the subprogram `factorial()` cited above, write a subprogram named `my_sin()` that accepts a value through its formal argument

list and returns the sine of x , using the Taylor series (shown next) with an accuracy of 0.0000000001.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

Hint: Keep in mind that x is in radians, and $\frac{x^1}{1!} = x$.

- iii. Write a subprogram named `degrees_to_rad()` that accepts an angle in degrees through its formal argument list and returns its radian equivalent. It is given that $2\pi = 360^\circ$.
- iv. Using the subprograms `my_sin()` and `degrees_to_rad()` cited above, write a PHP script that displays the sinus of all integers from 0° to 360° .

2. Do the following:

- i. Write a subprogram named `is_leap()` that accepts a year through its formal argument list and returns `true` or `false` depending on whether or not that year is a leap year.
- ii. Write a subprogram named `num_of_days()` that accepts a month and a year and returns the number of the days in that month. If that month is February and the year is a leap year, the subprogram must return the value of 29.

Hint: Use the subprogram `is_leap()` cited above.

- iii. Write a subprogram named `check_date()` that accepts a day, a month, and a year and returns `true` or `false` depending on whether or not that date is valid.
- iv. Using the subprograms cited above, write a PHP script that prompts the user to enter a date (a day, a month, and a year) and then calculates and displays the number of days that have passed between the beginning of the given year and the given date. Moreover, the script should validate data input and display an error message when the user enters any non-valid date.

3. In a computer game, players roll two dice. The player who gets the greatest sum of dice gets one point. After ten rolls, the player that wins is the one with the greatest sum of points. Do the following:

- i. Write a subprogram named `dice()` that returns a random integer between 1 and 6.
- ii. Using the subprogram `dice()` cited above, write a PHP script that prompts two players to enter their names and then each player consecutively “rolls” two dice. This process repeats ten times and the player that wins is the one with the greatest sum of points.

4. The LAV Car Rental Company has rented 40 cars, which are divided into three categories: hybrid, gas, and diesel. The company charges for a car according to the following table.

Days	Car Type		
	Gas	Diesel	Hybrid
1 – 5	\$24 per day	\$28 per day	\$30 per day
6 – 8	\$22 per day	\$25 per day	\$28 per day
9 and above	\$18 per day	\$21 per day	\$23 per day

Do the following:

- i. Write a subprogram named `get_choice()` that displays the following menu.
 - 1. Gas
 - 2. Diesel
 - 3. HybridThe subprogram then prompts the user to enter the type of the car (1, 2, or 3) and returns it to the caller.
- ii. Write a subprogram named `get_days()` that prompts the user to enter the total number of rental days and returns it to the caller.
- iii. Write a subprogram named `get_charge()` that accepts the type of the car (1, 2, or 3) and the total number of rental days through its formal argument list and then returns the amount of money to pay according to the previous table. Federal, state, and local taxes add a total of 10% to each bill.
- iv. Using the subprograms `get_choice()`, `get_days()` and `get_charge()` cited above, write a PHP script that prompts the user to enter all necessary information about the rented cars and then displays the following:
 - 1. for each car, the total amount to pay including taxes
 - 2. the total number of hybrid cars rented
 - 3. the total net profit the company gets after removing taxes

Please note that the rates are progressive.

- 5. TAM (Television Audience Measurement) is the specialized branch of media research dedicated to quantify and qualify television audience information. The LAV Television Audience Measurement Company counts the number of viewers of the main news program on each of 10 different TV channels. The company needs a software application in order to get some useful information. Do the following:
 - i. Write a subprogram named `get_data()` that prompts the user to enter into two arrays the names of the channels and the number of viewers of the main news program for each day of the week (Monday to Sunday). It then returns these arrays to the caller.

- ii. Write a subprogram `get_average()` that accepts a one-dimensional array of five numeric elements through its formal argument list and returns their average value.
 - iii. Using the subprograms `get_data()` and `get_average()` cited above, write a PHP script that prompts the user to enter the names of the channels and the number of viewers for each day of the week and then displays the following:
 - i. the name of the channels whose average viewer numbers on the weekend were at least 20% higher than the average viewer numbers during the rest of the week.
 - ii. the name of the channels (if any) that, from day to day, showed constantly increasing viewer numbers. If there is no such channel, a corresponding message must be displayed.
6. A public opinion polling company asks 300 citizens whether they have been hospitalized during the last year. Do the following:
 - i. Write a subprogram named `input_data()` that prompts the user to enter the citizen's SSN (Social Security Number) and their answer (Yes, No) into two arrays, `$SSNs` and `$answers`, respectively. The two arrays must be returned to the caller.
 - ii. Write a subprogram named `sort_arrays()` that accepts the arrays `$SSNs` and `$answers` through its formal argument list. It then sorts array `$SSNs` in ascending order using the selection sort algorithm. The subprogram must preserve the relationship between the elements of the two arrays.
 - iii. Write a subprogram named `search_array()` that accepts array `$SSNs` and an SSN through its formal argument list and then returns the index position of that SSN in the array. If the SSN is not found, a message "SSN not found" should be displayed and the value `-1` must be returned. Use the binary search algorithm.
 - iv. Write a subprogram named `count_answers()` that accepts the array `$answers` and an answer through its formal argument list. It then returns the number of times this answer exists in the array.
 - v. Using the three subprograms cited above, write a PHP script that prompts the user to enter the SSNs and the answers of the citizens. It should then prompt the user to enter an SSN and display the answer that the citizen with this SSN gave, as well as the percentage of citizens that gave the same answer. The script should then ask the user if he or she wishes to search for another SSN. If the answer is "Yes" the process should repeat; it should end otherwise.
7. Eight teams participate in a football tournament, and each team plays 12 games, one game each week. Do the following:

- i. Write a subprogram named `input_data()` that prompts the user to enter into two arrays the name of each team and the letter “W” for win, “L” for loss, or “T” for tie (draw) for each game. It then returns the arrays to the caller.
- ii. Write a subprogram named `display_result()` that prompts the user for a letter (W, L, or T) and then displays, for each team, the week number(s) in which the team won, lost, or tied respectively. For example, if the user enters “L”, the subprogram should search and display, for each team, the week numbers (e.g., week 3, week 14, and so on) in which the team lost the game.
- iii. Write a subprogram named `find_team()` that prompts the user to enter the name of a team and returns the index position of that team in the array. If the given team name does not exist, the value – 1 must be returned.
- iv. Using the three subprograms cited above, write a PHP script that prompts the user to enter the name of each team and the letter “W” for win, “L” for loss, or “T” for tie (draw) for each game. It should then prompt the user for a letter (W, L, or T) and display, for each team, the week number(s) in which the team won, lost, or tied respectively. Finally, the script should prompt the user to enter the name of a team. If the given team is found, the script should display the total number of points for this team and then prompt the user to enter the name of another team. This process must repeat as long as the user enters an existing team name. If given team name is not found, the message “Team not found” must be displayed and the script should end.

It is given that a win receives 3 points and a tie receives 1 point.

8. During the Cold War, messages were encrypted so that if the enemies intercepted them, they could not decrypt them without the decryption key. A very simple encryption algorithm is alphabetic rotation. The algorithm moves all letters N steps “up” in the alphabet, where N is the encryption key. For example, if the encryption key is 2, you can encrypt a message by replacing the letter A with the letter C, the letter B with the letter D, the letter C with the letter E, and so on. Do the following:
 - i. Write a subprogram `my_encrypt()` that accepts a message and an encryption key through its formal argument list and returns the encrypted message.
 - ii. Write a subprogram `my_decrypt()` that accepts an encrypted message and the decryption key through its formal argument list and returns the decrypted message.
 - iii. Write a subprogram named `display_menu()` that displays the following menu:
 1. Encrypt a message
 2. Decrypt a message

3. Exit

- iv. Using the three subprograms cited above, write a PHP script that displays the menu previously mentioned and then prompts the user to enter a choice (of 1, 2, or 3). If choice 1 is selected, the script should prompt the user to enter a message and an encryption key and then display the encrypted message. If choice 2 is selected, the script should prompt the user to enter an encrypted message and the decryption key and then display the decrypted message. The process should repeat as many times as the user wishes.

Assume that the user enters only lowercase letters for the message and a value between 1 and 26 for the encryption key.

Review Questions in “Subprograms”

Answer the following questions.

1. What is procedural programming?
2. What are the advantages of procedural programming?
3. What is meant by the term “spaghetti code”?
4. What is modular programming?
5. What is a subprogram? Name an example of a subprogram.
6. What is the general form of a PHP function?
7. How do you make a call to a function?
8. Describe the steps that are performed when the main code makes a call to a function.
9. What is the difference between a function and a procedure?
10. What is a void function?
11. What is the general form of a PHP procedure?
12. How do you make a call to a procedure?
13. Describe the steps that are performed when the main code makes a call to a procedure.
14. What is the formal argument list?
15. What is the actual argument list?
16. Can two subprograms use variables of the same name?
17. How long does a subprogram’s variable “live” in main memory?
18. How long does a main code’s variable “live” in main memory?
19. Can a subprogram call another subprogram? If yes, give some examples.
20. What does it mean to “pass an argument by value”?
21. What does it mean to “pass an argument by reference”?
22. Explain why you cannot pass an expression by reference. What can you do to overcome this limitation?
23. What is an optional argument?
24. What is the scope of a variable?
25. What happens when a variable has a local scope?
26. What happens when a variable has a global scope?
27. What is the difference between a local and a global variable?
28. What is recursion?
29. What are the three rules that all recursive algorithms must follow?

Index

A

`abs()`, 136, 198
absolute value, 198
absolute zero, 554
accumulate, 368, 518
accumulated, 368, 517, 518
actual argument list, 778, 791
add a breakpoint, 119
algorithm, 51
almost arbitrary initial value, 552, 688
alphanumeric, 71, 722
AND, 182, 258
antidiagonal, 619
application software, 29, 30
argument list, 778, 791
Aristides, 22
arithmetic operation, 183
arithmetic operator, 86, 184
array, 580
arrow, 53
ascending order, 703, 706
assign, 67
assigned, 84
asterisk-slash, 60
auxiliary array, 633, 635, 637, 639, 696,
 752, 756, 758

B

base case, 825
Beaufort Francis, 235
binary, 540
binary form, 540
binary search algorithm, 728, 741
BMI, 133, 786
Body Mass Index, 133, 786
Boolean, 71

Boolean expression, 180
Boolean value, 180
break, 229, 272, 444, 506, 543, 563
break out, 543
break out of the loop, 443, 444
breakpoint, 119
Brook Taylor, 571
bubble sort algorithm, 702, 703, 865
bug, 114
built-in, 772, 775, 782
by reference, 806
by value, 805
byref, 844

C

caller, 775, 789, 802
case decision structure, 228, 231
Celsius, 133
Central Processing Unit, 29
change of state, 825
character, 71
CLI, 37, 39
code, 31
code indentation, 280
code optimization, 251, 439
column, 580
column index, 608
command, 29, 31, 51, 57
Command Line Interface, 37
comment, 60
comparison operation, 183
comparison operator, 180, 184
compiled, 36
compiler, 31
complex Boolean expression, 182
complex mathematical expression, 146
compound assignment operator, 88,
 406

computer language, 30
computer program, 52
concatenation, 91
concatenation assignment, 91
consecutive ranges of values, 339
Const, 68
constant, 68, 594
control structure, 57
Corrado Böhm, 57
cos(), 137
cosine, 137
counter, 591
counter variable, 591, 738
CPU, 29
cross-platform, 39

D

data input, 52, 54, 80
data processing, 52
data validation, 319, 321, 530, 681
debugger, 116
debugging, 114
debugging step by step, 115
decision control structure, 57, 180
Decision symbol, 54, 372
declaration, 72
declare, 73
declare constant, 73
declare variable, 73
decrementing operator, 90, 406
default, 229, 239, 272
default argument value, 815
default precedence, 183
define, 73
definiteness, 51, 591
descending order, 709, 726
diamond, 372, 389
DIV, 154, 310
divisible, 319
divisor, 541
double quotes, 164
double slashes, 60, 91

do-while, 389
dual-alternative decision structure, 206

E

[echo](#), 77
effectiveness, 52
element, 581
end a loop, 443
End symbol, 53
endless iterations, 449
endless loop, 449
equal, 180
even, 211
execute, 30, 113
execution time, 265

F

factorial, 824
Fahrenheit, 130, 558
fgets, 81
Fibonacci, 570
Fibonacci sequence, 825
Finish Debugger Session, 118
finite, 51
finite sequence, 51
finiteness, 51
five-dimensional array, 583
flag, 445, 446, 565, 715, 732, 751, 754
flow of execution, 53, 116
flowchart, 53
for, 405
formal argument list, 778, 791
four-dimensional array, 581, 583
from inner to outer, 281, 476, 633, 638, 706
function, 136, 164, 772, 789

G

Giuseppe Jacopini, 57
global, 816

global scope, 816
Grace Hopper, 114
graphical method, 53
greater than, 180
greater than or equal to, 180
green arrow, 116

H

hardware, 29
heap sort algorithm, 702
Heron, 136
Heron's formula, 151, 536, 546
high-level, 37
high-level language, 31
hosting application, 36

I

IDE, 31, 41
if, 196
if-else, 206
if-elseif, 220
incoming off-page connector symbol,
 54
incrementing operator, 90, 406
indentation, 373
index, 165, 581, 588
index position, 581
indexed array, 582
infinite loop, 449
infinite number of times, 376, 394
inout, 845
input, 51
input device, 29
input value, 51
insertion sort algorithm, 702, 723, 726
instruction, 29, 31, 51
integer, 71, 321
integer division, 86
integer value, 137
Integrated Development Environment,
 31, 41

interpreted, 36
interpreter, 31, 37
intval(), 137, 153, 310
is_numeric(), 170, 321
iterate, 591
iterate endlessly, 450
iterate forever, 449
iterating through columns, 612
iterating through rows, 611, 614
iteration, 372, 389

J

JDK, 42

K

Kelvin, 130, 558
keyword, 56
keyword *break*, 229
keyword byref, 844
keyword *default*, 229, 239, 272
keyword global, 816
keyword *inout*, 845
KISS principle, 828

L

Law of Least Astonishment, 61
leap year, 353
left & right trimming, 167
left arrow, 66, 84
left gray margin, 119
left trimming, 166
Leibniz, 548
Leonardo Pisano Bigollo. *See Fibonacci*
less than, 180
less than or equal to, 180
line break, 79
linear equation, 331
linear search algorithm, 728, 729, 731
list of arguments, 775, 789
local scope, 816

logic error, 59, 114
logical complement, 182, 183
logical conjunction, 182, 183
logical disjunction, 182, 183
logical operation, 183
logical operator, 182, 183, 184
loop control structure, 57, 367
low-level language, 30
ltrim(), 166

M

machine language, 30, 37
Madhava, 548
Madhava–Leibniz series, 548
main code, 781, 782, 790, 801
main diagonal, 616, 620
main memory, 29, 30
mathematical expression, 146
mathematical function, 136
matrix, 580, 581
maximum, 335, 550, 686
merge sort algorithm, 702
mid-test loop structure, 489, 490
minimum, 335, 550, 686
MOD, 86
modified bubble sort algorithm, 702,
714
modular programming, 772
modularization, 57
module, 57, 772
modulus operator, 86
multidimensional array, 581, 582, 605
multiple-alternative decision structure,
219, 231
multiplication table, 477

N

negate, 188
negated operator, 188
negation, 182, 188
nested, 238

nested decision control structure, 238
nested loop, 425
NetBeans, 31, 37, 41
non-constant offset, 459
non-negative, 532, 546
non-positive, 395, 417, 536
NOT, 182, 189
not equal, 180
number of characters, 168
number of spaces per indent, 47

O

odd, 211
off-page connector symbol, 54
one-dimensional array, 581
open source, 36
operating system, 30
optional argument, 815
OR, 182, 259
outgoing off-page connector symbol, 54
output, 51
output device, 30
output value, 51

P

palindrome, 153, 356, 561
parenthesis, 87, 196
part of a string, 169
pass value, 775, 776, 790
passing arguments by reference, 805
passing arguments by value, 805
Personal Home Page, 36
PHP, 29, 31, 36
PHP project, 108
PHP script, 52
php.exe, 46
php.ini, 41
pi(), 137
pink rectangle, 120
POLA, 61
post-decrementing operator, 90

post-incrementing operator, 90
post-test loop structure, 389
pow(), 138, 140
power, 138
precedence rule, 87, 146, 183
pre-decrementing operator, 90
pre-defined process, 54
pre-incrementing operator, 90
preserving the relationship, 709
pre-test loop structure, 372
prime, 543, 544
Principle of Least Astonishment, 61
problem, 51
procedural programming, 771
procedure, 772, 789
Process symbol, 54
product, 378, 379, 516
program, 29, 30, 36
program counter, 116
programmer, 29, 52
progressive rates, 348, 867
property, 51
property of definiteness, 417, 519
property of finiteness, 394

Q

quadratic equation, 333
quicksort algorithm, 702
quotient, 153

R

RAM, 29
rand(), 138, 172
random, 138
Random Access Memory, 29
random word, 172
real, 71, 321
recursion, 823
red exclamation mark, 60, 115
region of interest, 188
remainder, 153

remove a breakpoint, 121
reserved word, 56
results output, 52, 54
return an array, 815
reversed number, 160
rhombus, 372
right trimming, 167
round, 138
round(), 138
row, 580
row index, 608
rtrim(), 167
run, 30
Russian multiplication algorithm, 538

S

scope of a variable, 816
script, 36
scripting language, 36
search algorithm, 728
secondary storage device, 29, 30
selection sort algorithm, 702, 720
semicolon, 59, 390
send value, 776, 790
sequence control structure, 57, 127
sequential search algorithm, 728
server-side, 36
sharp symbol, 60
shifting, 540
shifting operator, 541
simple Boolean expression, 180
sin(), 139
sine, 139
single quotes, 164
single-alternative decision structure, 196
slash-asterisk, 60
software, 30
software developer, 29
sort alphanumeric data, 722, 726
sort alphanumeric values, 708
sorting algorithm, 702

source code, 31
spaghetti code, 771, 818
sqrt(), 139, 140, 141
square array, 616, 620
square root, 138, 139
stand-alone, 36
STANDARD INPUT, 81
Start symbol, 53
statement, 29, 51, 57
STDIN, 81
step by step, 115
step into, 117
str_replace(), 167
string, 71, 164
string function, 164
string index, 166
string position, 168
string replacement, 167
strlen(), 168
strpos(), 168, 171
strtolower(), 168, 172
strtoupper(), 169, 345, 523, 536
structured code, 57
structured design, 57
structured programming, 57
sub-algorithm, 839
subproblem, 771
subprogram, 54, 771, 772
substr(), 169, 172
substring notation, 165
sum, 154, 377, 515
swap, 101, 102, 703, 865
[switch](#), 229, 239
syntax error, 59, 114
system software, 30

T

tab character, 79
tab stop, 79

tan(), 139
tangent, 139
Taylor series, 571
three main stages, 52
three parties, 52
three-dimensional array, 581, 582
to lowercase, 168
to uppercase, 169
trace table, 99, 197
trim(), 167
trimming, 166
truth table, 182
two-dimensional array, 581, 608

U

ultimate rule, 439, 521, 522, 523, 536
unit, 772
user, 52
user-defined function, 775, 782, 809
user-defined procedure, 810
user-friendly, 61

V

value assignment operator, 67, 84
variable, 66
visual representation, 53
void function, 789, 793

W

wavy red line, 114
[while](#), 372
William Shakespeare, 191

X

XAMPP, 37, 39

My Books

Available at
amazon

