

Applied C and C++ Programming

Alice E. Fischer
David W. Eggert
University of New Haven

Michael J. Fischer
Yale University

August 2018

Copyright ©2018

by **Alice E. Fischer, David W. Eggert, and Michael J. Fischer**

All rights reserved. This manuscript may be used freely by teachers and students in classes at the University of New Haven and at Yale University. Otherwise, no part of it may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the authors.

Contents

I	Introduction	1
1	Computers and Systems	3
1.1	The Physical Computer	3
	<i>Fig. 1.1:</i> Basic architecture of a modern computer.	3
1.1.1	The Processor	3
1.1.2	The Memory	4
	<i>Fig. 1.2:</i> Main memory in a byte-addressable machine.	5
	<i>Fig. 1.3:</i> The memory hierarchy.	5
1.1.3	Input and Output Devices	8
1.1.4	The Bus	9
1.1.5	Networks	9
	<i>Fig. 1.4:</i> A local network in a student lab.	10
1.2	The Operating System	10
1.3	Languages	12
1.3.1	Machine Language	12
1.3.2	Assembly Languages	13
1.3.3	High-Level Languages	13
1.4	What You Should Remember	15
1.4.1	Major Concepts	15
1.4.2	Vocabulary	15
1.5	Using Pencil and Paper	15
1.5.1	Self-Test Exercises	15
1.5.2	Using Pencil and Paper	15
2	Programs and Programming	17
2.1	What Is a Program?	17
	<i>Fig. 2.1:</i> A simple algorithm: Find an average.	17
2.1.1	The Simplest Program	18
	<i>Fig. 2.2:</i> A simple program: Hail and farewell.	18
2.2	The stages of program development.	19
2.2.1	Define the Problem	19
	<i>Fig. 2.3:</i> Problem specification: Find the average exam score.	20
2.2.2	Design a Test Plan	20
	<i>Fig. 2.4:</i> Test plan for exam average program.	20
2.2.3	Design a Solution	21
	<i>Fig. 2.5:</i> Pseudocode program sketch: Find an average.	21
2.3	The Development Environment	21
2.3.1	The Text Editor	21
	<i>Fig. 2.6:</i> The stages of program translation.	22
2.3.2	The Translator	22

2.3.3	Linkers	23
2.4	Source Code Construction	23
2.4.1	Create a Source File	23
2.4.2	Coding.	24
	Fig. 2.7: A C program.	24
2.4.3	Translate the Program and Correct Errors	25
2.5	Program Execution and Testing	26
2.5.1	Execute the Program	26
	Fig. 2.8: Memory for constants.	26
	Fig. 2.9: Testing a program.	26
	Fig. 2.10: Memory during program execution.	27
2.5.2	Test and Verify	28
2.6	What You Should Remember	28
2.6.1	Major Concepts	28
2.6.2	The moral of this story.	29
2.6.3	Vocabulary	29
2.7	Exercises	30
2.7.1	Self-Test Exercises	30
2.7.2	Using Pencil and Paper	30
2.7.3	Using the Computer	31
3	The Core of C	33
3.1	The Process of Compilation	33
	Fig. 3.1: The stages of compilation.	33
3.2	The Parts of a Program	34
3.2.1	Terminology.	34
	Fig. 3.2: How to lay out a simple program.	34
3.2.2	The main() program.	35
	Fig. 3.3: Words in C.	35
3.3	An Overview of Variables, Constants, and Expressions	36
3.3.1	Values and Storage Objects.	36
3.3.2	Variables	36
	Fig. 3.4: Syntax for declarations.	37
	Fig. 3.5: Simple declarations.	38
3.3.3	Constants and Literals	39
	Fig. 3.6: Using constants.	40
3.3.4	Names and Identifiers	40
3.3.5	Arithmetic and Formulas	41
3.4	Simple Input and Output	42
3.4.1	Formats.	42
3.5	A Program with Calculations	44
	Fig. 3.7: Grapefruits and gravity.	44
3.6	The Flow of Control	46
	Fig. 3.8: A complete flow diagram of a simple program.	47
	Fig. 3.9: Diagram of the grapefruits and gravity program.	47
3.7	Asking Questions: Conditional Statements	48
3.7.1	The Simple <code>if</code> Statement	48
	Fig. 3.10: Asking a question.	48
3.7.2	The <code>if...else</code> Statement	50
	Fig. 3.11: Testing the limits.	50
	Fig. 3.12: Flow diagram for Testing the limits.	50
3.7.3	Which <code>if</code> Should Be Used?	53

3.7.4	Options for Syntax and Layout (Optional Topic)	53
	Fig. 3.13: The <code>if</code> statement with and without braces.	53
3.8	Loops and Repetition	54
3.8.1	A Counting Loop	54
	Fig. 3.14: Countdown.	54
3.8.2	An Input Data Validation Loop	56
	Fig. 3.15: Input validation using <code>while</code> .	56
3.9	An Application	58
	Fig. 3.16: Problem specification: Gas prices.	58
	Fig. 3.17: Test plan: Gas prices.	59
	Fig. 3.18: Problem solution: Gas prices.	60
3.10	What You Should Remember	61
3.10.1	Major Concepts	61
3.10.2	Programming Style	63
3.10.3	Sticky Points and Common Errors	63
3.10.4	New and Revisited Vocabulary	64
3.10.5	Where to Find More Information	64
3.11	Exercises	65
3.11.1	Self-Test Exercises	65
3.11.2	Using Pencil and Paper	66
3.11.3	Using the Computer	68
	Fig. 3.19: Sketch for the heat transfer program.	68
	Fig. 3.20: Sketch for the temperature conversion program.	68
	Fig. 3.21: Problem specification: Gasket area.	70
	Fig. 3.22: Flow diagram for the gasket area program.	70

II Computation 73

4	Expressions	75
4.1	Expressions and Parse Trees	75
4.1.1	Operators and Arity	75
4.1.2	Precedence and Parentheses	76
	Fig. 4.1: Arithmetic operators.	76
4.1.3	Parsing and Parse Trees	77
	Fig. 4.2: Applying precedence.	77
	Fig. 4.3: Applying associativity.	77
	Fig. 4.4: Applying the parsing rules.	78
4.2	Arithmetic, Assignment, and Combination Operators	79
	Fig. 4.5: Assignment and arithmetic combinations.	79
	Fig. 4.6: Parse tree for an assignment combination.	79
	Fig. 4.7: Arithmetic combinations.	80
4.3	Increment and Decrement Operators	81
	Fig. 4.8: Increment and decrement operators.	81
4.3.1	Parsing Increment and Decrement Operators	81
	Fig. 4.9: Increment and decrement trees.	81
4.3.2	Prefix versus Postfix Operators	81
4.3.3	Mixing increment or decrement with other operators.	82
	Fig. 4.10: Evaluating an increment expression.	82
4.4	The <code>sizeof</code> operator.	83
	Fig. 4.11: The size of things.	83
4.5	Relational Operators	84

	<i>Fig. 4.12:</i> Relational operators.	84
4.5.1	True and False	84
4.5.2	The semantics of comparison.	85
4.6	Logical Operators	86
4.6.1	Truth-valued operators.	86
	<i>Fig. 4.13:</i> Truth table for the logical operators.	86
4.6.2	Parse Trees for Logical Operators	86
4.6.3	Lazy Evaluation	87
	<i>Fig. 4.14:</i> Precedence of the logical operators.	87
	<i>Fig. 4.15:</i> Parsing logical operators.	87
	<i>Fig. 4.16:</i> Lazy truth table for logical-AND.	87
	<i>Fig. 4.17:</i> Lazy evaluation of logical-AND.	87
	<i>Fig. 4.18:</i> Lazy truth table for logical-OR.	87
	<i>Fig. 4.19:</i> Lazy evaluation of logical-OR.	88
4.7	Integer Operations	89
4.7.1	Integer Division and Modulus	89
	<i>Fig. 4.20:</i> The modulus operation is cyclic.	90
4.7.2	Applying Integer Division and Modulus	90
	<i>Fig. 4.21:</i> Visualizing the modulus operator.	91
	<i>Fig. 4.22:</i> Positional notation and base conversion.	91
	<i>Fig. 4.23:</i> Number conversion.	91
	<i>Fig. 4.24:</i> A flow diagram for the base conversion.	91
4.8	Techniques for Debugging	94
4.8.1	Using Assignments and Printouts to Debug	94
	<i>Fig. 4.25:</i> Problem specification: Ice cream for the picnic.	94
	<i>Fig. 4.26:</i> How much ice cream?	94
4.8.2	Case Study: Using a Parse Tree to Debug	96
	<i>Fig. 4.27:</i> Problem specification: Computing resistance.	96
	<i>Fig. 4.28:</i> Computing resistance.	98
	<i>Fig. 4.29:</i> Finding an expression error.	99
	<i>Fig. 4.30:</i> Parsing the corrected expression.	99
4.9	What You Should Remember	100
4.9.1	Major Concepts	100
4.9.2	Programming Style	101
4.9.3	New and Revisited Vocabulary	101
	<i>Fig. 4.31:</i> Difficult aspects of C operators.	101
4.9.4	Sticky Points and Common Errors	101
4.9.5	Where to Find More Information	101
4.10	Exercises	102
4.10.1	Self-Test Exercises	102
4.10.2	Using Pencil and Paper	104
4.10.3	Using the Computer	105
5	Using Functions and Libraries	109
5.1	Libraries	109
5.1.1	Standard Libraries	110
5.1.2	Other Libraries	110
5.1.3	Using Libraries	110
	<i>Fig. 5.1:</i> A library is like an iceberg: only a small part is visible.	110
	<i>Fig. 5.2:</i> Form of a simple function prototype.	111
5.2	Function Calls	112
	<i>Fig. 5.3:</i> Form of a simple function call.	112

	Fig. 5.4: Calling library functions.	112
5.2.1	Call Graphs	114
	Fig. 5.5: Call graph for Figure 5.4: Calling library functions.	115
5.2.2	Math Library Functions	115
	Fig. 5.6: Functions in the math library.	115
	Fig. 5.7: Rounding down: <code>floor()</code>	116
	Fig. 5.8: Rounding up: <code>ceil()</code>	116
	Fig. 5.9: Rounding to the nearest integer: <code>rint()</code>	117
	Fig. 5.10: Truncation via assignment.	117
5.2.3	Other Important Library Functions	117
	Fig. 5.11: A few functions in other libraries.	118
5.3	Programmer-Defined Functions	118
5.3.1	Function syntax.	119
5.3.2	Defining Void→Void Functions	119
	Fig. 5.12: Using void→void functions.	119
	Fig. 5.13: Printing a banner on your output.	119
	Fig. 5.14: A call graph for the beep program.	119
5.3.3	Returning Results from a Function	122
5.3.4	Arguments and Parameters	122
5.3.5	Defining a Double→Double Function	123
	Fig. 5.15: The grapefruit returns.	123
	Fig. 5.16: Call graph for the grapefruit returns.	125
	Fig. 5.17: Flow diagram for the grapefruit returns.	125
	Fig. 5.18: Definition of the <code>drop()</code> function.	125
5.4	Organization of a Module	127
	Fig. 5.19: Functions, prototypes, and calls.	127
	Fig. 5.20: Function call graph with two levels of calls.	127
	Fig. 5.21: A function is a black box.	129
5.5	Application: Numerical Integration by Summing Rectangles	130
	Fig. 5.22: The area under a curve.	130
	Fig. 5.23: Integration by summing rectangles.	130
5.6	Functions with More Than One Parameter	132
5.6.1	Function Syntax: Two Parameters	132
	Fig. 5.24: Using functions with two parameters and return values.	133
	Fig. 5.25: Functions with two parameters and return values.	133
5.7	Application: Generating “Random” Numbers	135
5.7.1	Pseudo-Random Numbers	135
5.7.2	How Good Is the Standard Random Number Generator?	136
	Fig. 5.26: Generating random numbers.	136
5.8	Application: A Guessing Game	138
5.8.1	Strategy	138
5.8.2	Playing the Game	139
	Fig. 5.27: Halving the range.	139
	Fig. 5.28: Can you guess my number?	139
	Fig. 5.29: Guessing a number.	139
5.9	What You Should Remember	142
5.9.1	Major Concepts	142
5.9.2	Local Libraries and Header Files	143
5.9.3	The Order of the Parts of a Program	144
5.9.4	Programming Style	145
5.9.5	Sticky Points and Common Errors	145
5.9.6	Where to Find More Information	146

5.9.7	New and Revisited Vocabulary	146
5.10	Exercises	147
5.10.1	Self-Test Exercises	147
5.10.2	Using Pencil and Paper	148
5.10.3	Using the Computer	149
6	More Repetition and Decisions	153
6.1	New Loops	153
6.1.1	The for Loop	153
	Fig. 6.1: The for statement is a shorthand form of a while statement.	153
	Fig. 6.2: A simple program with a for statement.	153
	Fig. 6.3: A flow diagram for the for statement.	154
	Fig. 6.4: Diagrams of corresponding while and for loops.	154
6.1.2	The do...while Loop	156
	Fig. 6.5: A flow diagram for the do...while statement.	156
6.1.3	Other Control Statements	156
	Fig. 6.6: Using break with while and do	157
	Fig. 6.7: Using continue with while and do	157
	Fig. 6.8: Using continue with for	158
6.1.4	Defective Loops	158
	Fig. 6.9: No update and no exit: Two defective loops.	159
6.2	Applications of Loops	159
6.2.1	Sentinel Loops	160
	Fig. 6.10: A cash register program.	160
6.2.2	Query Loops	161
	Fig. 6.11: Form of a query loop.	161
	Fig. 6.12: Repeating a calculation.	162
	Fig. 6.13: The work() and drop() functions.	162
	Fig. 6.14: Flow diagram for repeating a process.	162
6.2.3	Counted Loops	164
	Fig. 6.15: Summing a series.	164
6.2.4	Input Validation Loops	166
	Fig. 6.16: Input validation using a while statement.	166
6.2.5	Nested Loops	167
	Fig. 6.17: Printing a table with nested for loops.	167
	Fig. 6.18: Flow diagram for the multiplication table program.	167
6.2.6	Delay Loops	169
	Fig. 6.19: Using a delay loop.	169
	Fig. 6.20: Delaying progress, the delay() function.	169
6.2.7	Flexible-Exit Loops	171
	Fig. 6.21: A structured loop with break	171
	Fig. 6.22: Input validation loop using a for statement.	171
6.2.8	Counted Sentinel Loops	172
	Fig. 6.23: Skeleton of a counted sentinel loop.	172
	Fig. 6.24: Breaking out of a loop.	172
	Fig. 6.25: Avoiding a break-out.	172
6.2.9	Search Loops	174
	Fig. 6.26: Skeleton of a search loop.	174
6.3	The switch Statement	174
6.3.1	Syntax and Semantics	174
	Fig. 6.27: Problem specification: Wire gauge adequacy evaluation.	175
	Fig. 6.28: Diagrams for a nested conditional and a corresponding switch	176

6.3.2	A <code>switch</code> Application	177
	Fig. 6.29: Using a <code>switch</code>	177
6.4	Search Loop Application: Guess My Number	179
	Fig. 6.30: Problem specification: Guess my number.	179
	Fig. 6.31: An input-driven search loop.	179
	Fig. 6.32: A counted sentinel loop.	179
6.5	What You Should Remember	181
6.5.1	Major Concepts	181
6.5.2	Programming Style	182
6.5.3	Sticky Points and Common Errors	182
6.5.4	Where to Find More Information	183
6.5.5	New and Revisited Vocabulary	183
6.6	Exercises	184
6.6.1	Self-Test Exercises	184
6.6.2	Using Pencil and Paper	185
6.6.3	Using the Computer	187

III Basic Data Types 191

7 Using Numeric Types 193

7.1	Integer Types	193
	Fig. 7.1: Names for integer types.	193
7.1.1	Signed and Unsigned Integers	194
7.1.2	Short and long integers.	194
	Fig. 7.2: ISO C integer representations.	194
	Fig. 7.3: Integer literals in base 10.	195
7.2	Floating-Point Types in C	195
	Fig. 7.4: IEEE floating-point types.	195
	Fig. 7.5: Floating-point literal examples	196
7.3	Reading and Writing Numbers	196
7.3.1	Integer Input	197
	Fig. 7.6: Integer conversion specifications.	197
7.3.2	Integer Output	197
	Fig. 7.7: Incorrect conversions produce garbabe output.	197
7.3.3	Floating-Point Input	198
7.3.4	Floating-Point Output	198
	Fig. 7.8: Basic floating-point conversion specifications.	199
	Fig. 7.9: The <code>%f</code> output conversion.	199
	Fig. 7.10: The <code>%e</code> output conversion.	199
	Fig. 7.11: Sometimes <code>%g</code> output looks like an integer.	199
	Fig. 7.12: Sometimes <code>%g</code> looks like <code>%e</code>	199
	Fig. 7.13: For tiny numbers, <code>%g</code> looks like <code>%e</code>	199
	Fig. 7.14: Sometimes <code>%g</code> looks like <code>%f</code>	199
	Fig. 7.15: Output conversion specifiers.	200
7.3.5	One Number may Appear in Many Ways	200
7.4	Mixing Types in Computations	202
7.4.1	Basic Type Conversions	202
	Fig. 7.16: Converting a <code>float</code> to a <code>double</code>	202
7.4.2	Type Casts and Coercions	203
	Fig. 7.17: Kinds of casts.	203
	Fig. 7.18: Rounding and truncation.	204

	Fig. 7.19: Type coercion.	205
7.4.3	Diagramming Conversions	206
	Fig. 7.20: Diagramming a cast and a coercion.	206
	Fig. 7.21: Expressions with casts and coercions.	206
	Fig. 7.22: Problem specification: computing resistance.	207
7.4.4	Using Type Casts to Avoid Integer Division Problems	207
	Fig. 7.23: Computing resistance.	208
7.5	The Trouble with Numbers	209
7.5.1	Overflow	210
	Fig. 7.24: Overflow and wrap with a four bit signed integer.	210
	Fig. 7.25: Computing $N!$	211
7.5.2	Underflow	213
	Fig. 7.26: Floating-point underflow.	213
7.5.3	Orders of Magnitude	214
7.5.4	Not a Number	214
	Fig. 7.27: Calculation order matters.	215
7.5.5	Representational Error	215
7.5.6	Making Meaningful Comparisons	216
	Fig. 7.28: An approximate comparison.	216
	Fig. 7.29: Comparing floats for equality.	216
7.5.7	Application: Cruise Control	216
	Fig. 7.30: Problem specification: Cruise control.	216
	Fig. 7.31: Cruise control.	216
7.6	What You Should Remember	219
7.6.1	Major Concepts	219
7.6.2	Sticky Points and Common Errors	221
	Fig. 7.32: Casts and conversions in C.	221
7.6.3	Programming Style	222
7.6.4	New and Revisited Vocabulary	223
7.6.5	Where to Find More Information	224
7.7	Exercises	224
7.7.1	Self-Test Exercises	224
7.7.2	Pencil and Paper	227
7.7.3	Using the Computer	229
8	Character Data	233
8.1	Representation of Characters	233
	Fig. 8.1: The case bit in ASCII.	233
8.1.1	Character Types in C	234
8.1.2	The Different Interpretations	234
	Fig. 8.2: Character types.	234
8.1.3	Character Literals	234
	Fig. 8.3: Writing character constants.	234
	Fig. 8.4: Useful predefined escape sequences.	234
8.2	Input and Output with Characters	235
8.2.1	Character Input	235
8.2.2	Character Output	236
	Fig. 8.5: Printing the ASCII codes.	236
8.2.3	Using the I/O Functions	237
	Fig. 8.6: Character input and output.	237
8.2.4	Other ways to skip whitespace.	239
	Fig. 8.7: A function for skipping whitespace.	239

8.3	Operations on Characters	240
8.3.1	Characters Are Very Short Integers	240
	Fig. 8.8: Character operations.	240
8.3.2	Assignment	240
8.3.3	Comparing Characters	241
8.3.4	Character Arithmetic	241
8.3.5	Other Character Functions	241
8.4	Character Application: An Improved Processing Loop	242
	Fig. 8.9: Improving the workmaster.	242
	Fig. 8.10: Using characters in a switch.	242
8.5	What You Should Remember	245
8.5.1	Major Concepts	245
8.5.2	Programming Style	246
8.5.3	Sticky Points and Common Errors	246
8.5.4	New and Revisited Vocabulary	246
8.6	Exercises	246
8.6.1	Self-Test Exercises	246
8.6.2	Using Pencil and Paper	247
8.6.3	Using the Computer	249
9	Program Design	253
9.1	Modular Programs	253
9.2	Communication Between Functions	253
9.2.1	The Function Interface	254
	Fig. 9.1: Information flow in <code>pow2()</code>	254
9.2.2	Parameter Passing	255
	Fig. 9.2: The run-time stack.	255
9.3	Parameter Type Checking	256
	Fig. 9.3: The declared prototype controls all.	256
	Fig. 9.4: The importance of parameter order.	257
9.4	Data Modularity	258
	Fig. 9.5: Gas models before global elimination.	259
	Fig. 9.6: A call graph for the CO gas program.	259
	Fig. 9.7: Functions for gas models before global elimination.	259
9.4.1	Scope and Visibility	259
9.4.2	Global and Local Names	261
	Fig. 9.8: Name scoping in gas models program, before global elimination.	261
9.4.3	Eliminating Global Variables	263
	Fig. 9.9: Eliminating the global variable.	263
	Fig. 9.10: Functions for gas models after global elimination.	263
9.5	Program Design and Construction	265
9.5.1	The Process	265
	Fig. 9.11: Problem specification and test plan: fin temperature.	265
9.5.2	Applying the Process: Stepwise Development of a Program	266
	Fig. 9.12: First draft of <code>main()</code> for the fin program, with a function stub.	267
	Fig. 9.13: Fin program: First draft of <code>print_table()</code> function.	269
	Fig. 9.14: Temperature along a cooling fin.	272
9.6	What You Should Remember	272
9.6.1	Major Concepts	272
9.6.2	Programming Style	274
9.6.3	Sticky Points and Common Errors	274
9.6.4	New and Revisited Vocabulary	275

9.7	Exercises	275
9.7.1	Self-Test Exercises	275
9.7.2	Using Pencil and Paper	277
9.7.3	Using the Computer	278
10	An Introduction to Arrays	283
10.1	Arrays	283
10.1.1	Array Declarations and Initializers	284
	Fig. 10.1: Declaring an array of five floats.	284
	Fig. 10.2: An initialized array of short <code>ints</code>	284
	Fig. 10.3: Length and initializer options.	284
	Fig. 10.4: Initializing character arrays.	285
10.1.2	The Size of an Array	285
	Fig. 10.5: The size of an array.	286
10.1.3	Accessing Arrays	286
	Fig. 10.6: Simple subscripts.	287
	Fig. 10.7: Computed subscripts.	287
	Fig. 10.8: Subscript demo, the magnitude of a vector in 3-space.	288
10.1.4	Subscript Out-of-Range Errors	289
10.2	Using Arrays	289
10.2.1	Array Input	289
	Fig. 10.9: Filling an array with data.	289
10.2.2	Walking on Memory	290
	Fig. 10.10: Walking on memory.	290
	Fig. 10.11: Before and after walking on memory.	290
10.3	Parallel Arrays	292
	Fig. 10.12: Problem specifications: Exam averages.	292
	Fig. 10.13: Using parallel arrays.	292
	Fig. 10.14: Parallel arrays can represent a table.	292
10.4	Array Arguments and Parameters	295
10.5	An Array Application: Prime Numbers	296
	Fig. 10.15: Problem specifications: A table of prime numbers.	296
	Fig. 10.16: Calculating prime numbers.	296
	Fig. 10.17: Functions for the prime number program.	298
10.6	Searching an Array	300
	Fig. 10.18: Problem specifications: Recording bill payments.	300
	Fig. 10.19: Main program for sequential search.	300
	Fig. 10.20: Input and output functions for parallel arrays.	303
	Fig. 10.21: Sequential search of a table.	304
	Fig. 10.22: Searching without a second return statement.	304
10.7	The Maximum Value in an Array	306
	Fig. 10.23: Who owes the most? Main program for finding the maximum.	306
	Fig. 10.24: Finding the maximum value.	306
	Fig. 10.25: The maximum algorithm, step by step.	306
10.8	Sorting by Selection	307
	Fig. 10.26: The selection sort algorithm, step by step.	307
	Fig. 10.27: Problem specifications: Sorting the Billing Records	309
10.8.1	The Main Program	309
	Fig. 10.28: Call chart for selection sort.	310
10.8.2	Developing the <code>sort_data()</code> Function	310
	Fig. 10.29: Main program for selection sort.	310
	Fig. 10.30: Sorting by selecting the maximum.	311

<i>Fig. 10.31:</i> Input and output for selection sort.	311
10.9 What You Should Remember	312
10.9.1 Major Concepts	312
10.9.2 Programming Style	313
10.9.3 Sticky Points and Common Errors	314
10.9.4 Where to Find More Information	315
10.9.5 New and Revisited Vocabulary	315
10.10 Exercises	316
10.10.1 Self-Test Exercises	316
10.10.2 Using Pencil and Paper	318
10.10.3 Using the Computer	319
11 An Introduction to Pointers	323
11.1 A First Look at Pointers	323
11.1.1 Pointer Values Are Addresses	323
<i>Fig. 11.1:</i> A pointer and its referent.	323
<i>Fig. 11.2:</i> Pointer diagrams.	323
<i>Fig. 11.3:</i> Declaring a pointer variable.	323
<i>Fig. 11.4:</i> Initializing a pointer variable.	324
11.1.2 Pointer Operations	325
<i>Fig. 11.5:</i> Pointers in memory.	325
<i>Fig. 11.6:</i> Pointer operations.	326
11.2 Call by Value/Address	327
11.2.1 Address Arguments	328
11.2.2 Pointer Parameters	328
<i>Fig. 11.7:</i> Call by value/address.	328
<i>Fig. 11.8:</i> A swap function with an error.	331
<i>Fig. 11.9:</i> Seeing the difference.	331
<i>Fig. 11.10:</i> A swap function that works.	331
11.2.3 A More Complex Example	331
11.3 Application: Statistical Measures	332
<i>Fig. 11.11:</i> Problem specifications: Statistics.	332
<i>Fig. 11.12:</i> Mean and standard deviation.	332
<i>Fig. 11.13:</i> Call graph for the mean and standard deviation program.	332
<i>Fig. 11.14:</i> The <code>get_data()</code> and <code>stats()</code> functions.	332
11.3.1 Summary: Returning Results from a Function	336
11.4 What You Should Remember	336
11.4.1 Major Concepts	336
11.4.2 Programming Style	337
11.4.3 Sticky Points and Common Errors	337
11.4.4 Where to Find More Information	338
11.4.5 New and Revisited Vocabulary	338
11.5 Exercises	338
11.5.1 Self-Test Exercises	338
11.5.2 Using Pencil and Paper	340
11.5.3 Using the Computer	342

IV	Representing Data	345
12	Strings	347
12.1	String Representation	347
12.1.1	String Literals	348
	Fig. 12.1: String literals.	348
	Fig. 12.2: Quotes and comments.	348
12.1.2	A String Is a Pointer to an Array	349
	Fig. 12.3: Three kinds of nothing.	349
	Fig. 12.4: Implementation of a cstring variable.	349
	Fig. 12.5: Selecting the appropriate answer.	349
12.1.3	Declare an Array to Get a String	350
	Fig. 12.6: An array of characters.	350
12.1.4	Array vs. String	351
	Fig. 12.7: Array vs. string.	351
	Fig. 12.8: A character array and a string.	351
	Fig. 12.9: C++ strings.	351
12.1.5	C++ Strings	352
12.2	C String I/O	352
	Fig. 12.10: String literals and string output in C.	352
	Fig. 12.11: String literals and string output in C++.	352
12.2.1	String Output	352
12.2.2	String Input in C	356
	Fig. 12.12: You cannot store an input string in a pointer.	356
	Fig. 12.13: You need an array to store an input string.	356
12.2.3	String Input in C++	358
12.2.4	Guidance on Input	359
12.3	String Functions	359
12.3.1	Strings as Parameters	359
	Fig. 12.14: A string parameter in C.	359
	Fig. 12.15: A string parameter in C++.	359
12.3.2	The String Library	360
	Fig. 12.16: The size of a C string.	362
	Fig. 12.17: The size of a C++ string.	362
	Fig. 12.18: Do not use == with C strings (but it works in C++).	363
	Fig. 12.19: Computing the length of a string in C.	363
	Fig. 12.20: Possible implementation of <code>strcmp()</code>	363
	Fig. 12.21: Searching for a character or a substring in C.	365
	Fig. 12.22: Search, copy and concatenate in C.	366
	Fig. 12.23: Search, copy and concatenate in C++.	366
12.4	Arrays of Strings	367
	Fig. 12.24: A ragged array.	367
12.4.1	The Menu Data Structure	368
12.4.2	An Example: Selling Ice Cream	368
	Fig. 12.25: Selecting ice cream from a menu.	368
	Fig. 12.26: A menu-handling function.	369
	Fig. 12.27: Buying a Cone in C++.	369
	Fig. 12.28: Using an invalid subscript.	370
12.5	String Processing Applications	373
12.5.1	Password Validation	373
	Fig. 12.29: Password validation: Comparing strings.	373
	Fig. 12.30: Password validation: Comparing strings.	373

12.5.2	The <code>menu_c()</code> Function	375
	Fig. 12.31: The <code>menu_c()</code> function.	376
	Fig. 12.32: <code>menu_c</code> in C++.	376
12.5.3	Menu Processing and String Parsing	377
	Fig. 12.33: Flow diagram for Figure 12.34.	377
	Fig. 12.34: Magic memo maker in C.	377
	Fig. 12.35: Magic memo maker in C++.	377
	Fig. 12.36: Using a string variable with a menu.	378
	Fig. 12.37: Composing and printing the memo in C.	382
	Fig. 12.38: Composing and printing the memo in C++.	382
12.6	What You Should Remember	384
12.6.1	Major Concepts	384
12.6.2	Programming Style	385
12.6.3	Sticky Points and Common Errors	385
12.6.4	New and Revisited Vocabulary	386
12.6.5	Where to Find More Information	387
13	Enumerated and Structured Types	389
13.1	Enumerated Types	389
13.1.1	Enumerations	389
	Fig. 13.1: The form of an <code>enum</code> declaration in C and C++.	390
	Fig. 13.2: Four enumerations in C and again in C++.	390
	Fig. 13.3: Using an enumeration to name errors.	391
13.1.2	Printing Enumeration Codes	391
13.1.3	Returning Multiple Function Results	392
	Fig. 13.4: Memory for the quadratic root program.	392
	Fig. 13.5: Solving a quadratic equation in C.	392
	Fig. 13.6: Solving a quadratic equation in C++.	394
	Fig. 13.7: The quadratic root function for both C and C++.	394
13.2	Structures in C	397
	Fig. 13.8: Modern syntax for a C <code>struct</code> declaration.	397
	Fig. 13.9: A <code>struct</code> type declaration in C.	398
	Fig. 13.10: Declaring and initializing a structure.	398
13.3	Operations on Structures	399
13.3.1	Structure Operations	399
	Fig. 13.11: Set and use a pointer to a structure.	400
	Fig. 13.12: Access one member of a structure.	400
	Fig. 13.13: Structure assignment.	400
13.3.2	Using Structures with Functions	400
	Fig. 13.14: Returning a structure from a function.	400
	Fig. 13.15: Call by value with a structure.	401
	Fig. 13.16: Call by address with a structure.	402
	Fig. 13.17: An array of structures.	403
13.3.3	Arrays of Structures	403
	Fig. 13.18: Comparing two structures in C.	405
	Fig. 13.19: The <code>print_inventory</code> function.	405
	Fig. 13.20: Declarations for the lumber program.	405
	Fig. 13.21: Structure operations: the whole program in C.	405
13.3.4	Comparing Two Structures	405
13.3.5	Putting the Pieces Together	406
13.4	Structures and Classes in C++	408
13.4.1	Definitions and Conventions.	408

13.4.2	Function Members of a Class	409
13.4.3	Encapsulation	410
13.4.4	Instantiation and Memory Management	411
13.5	The Lumber Program in C++	411
	Fig. 13.22: The LumberT Class in C++	412
	Fig. 13.23: The Lumber program in C++	412
	Fig. 13.24: The Lumber functions in C++	414
13.6	Application: Points in a Rectangle	416
	Fig. 13.25: A rectangle on the xy -plane.	416
	Fig. 13.26: Problem specifications: Points in a rectangle.	416
	Fig. 13.27: The test plan for points in a rectangle.	417
	Fig. 13.28: Two structured types.	417
	Fig. 13.29: Header file for points in a rectangle.	417
13.6.1	The program: Points in a Rectangle	418
	Fig. 13.30: Main program for points in a rectangle.	420
	Fig. 13.31: Making a rectangle.	421
	Fig. 13.32: Testing points.	422
	Fig. 13.33: Comparing two points.	423
	Fig. 13.34: Point location.	423
	Fig. 13.35: Main function in C++ for points and rectangles.	425
13.7	Points in a Rectangle written in C++	425
	Fig. 13.36: The C++ class declarations for points and rectangles.	427
	Fig. 13.37: C++ functions for points.	428
	Fig. 13.38: C++ functions for rectangles.	428
13.8	What You Should Remember	430
13.8.1	Major Concepts	430
13.8.2	Programming Style	431
13.8.3	Sticky Points and Common Errors	432
13.8.4	New and Revisited Vocabulary	433
13.8.5	Where to Find More Information	433
14	Streams and Files	435
14.1	Streams and Buffers	435
14.1.1	Stream I/O	435
	Fig. 14.1: A stream carries data.	436
	Fig. 14.2: Standard streams.	436
	Fig. 14.3: The three default streams.	436
14.1.2	Buffering	437
	Fig. 14.4: An input buffer is a window on the data.	437
14.1.3	Formatted and Unformatted I/O.	438
14.2	Programmer-Defined Streams	439
14.2.1	Defining and Using Streams	439
	Fig. 14.5: Opening streams in C++.	439
	Fig. 14.6: Programmer-defined streams.	440
14.2.2	File Management Errors	441
14.3	Stream Output	441
	Fig. 14.7: Specifications: Creating a data table.	441
	Fig. 14.8: Writing a file of voltages.	441
14.4	Stream Input	443
14.4.1	Input Functions	443
14.4.2	Detecting End of File	444
14.4.3	Reading Data from a File	445

<i>Fig. 14.9:</i>	Reading a data file.	445
14.4.4	Using Unformatted I/O	447
<i>Fig. 14.10:</i>	Copying a file.	447
14.4.5	Reading an Entire File at Once.	448
<i>Fig. 14.11:</i>	Reading an entire file.	448
14.5	Stream Errors	449
14.5.1	A Missing Newline	450
14.5.2	An Illegal Input Character	450
14.6	File Application: Random Selection Without Replacement	451
<i>Fig. 14.12:</i>	Problem specifications: A quiz.	452
<i>Fig. 14.13:</i>	An elemental quiz.	452
<i>Fig. 14.14:</i>	Call chart for the Element Quiz	452
<i>Fig. 14.15:</i>	The Quiz class declaration.	453
<i>Fig. 14.16:</i>	The Element class declaration.	454
<i>Fig. 14.17:</i>	The implementation of the Element class.	455
<i>Fig. 14.18:</i>	The Quiz Constructor.	456
<i>Fig. 14.19:</i>	The doQuiz function.	458
<i>Fig. 14.20:</i>	The array after one question.	460
<i>Fig. 14.21:</i>	The array after three questions.	460
<i>Fig. 14.22:</i>	The array after five questions.	460
<i>Fig. 14.23:</i>	The array after seven questions.	460
14.7	What You Should Remember	461
14.7.1	Major Concepts	461
14.7.2	Programming Style	463
14.7.3	Sticky Points and Common Errors	463
14.7.4	New and Revisited Vocabulary	464
14.7.5	Where to Find More Information	464
15	Calculating with Bits	465
15.1	Number Representation and Conversion	465
15.1.1	Hexadecimal Notation	465
<i>Fig. 15.1:</i>	Hexadecimal numeric literals.	466
15.1.2	Number Systems and Number Representation	466
<i>Fig. 15.2:</i>	Place values.	466
15.1.3	Signed and Unsigned Integers	467
<i>Fig. 15.3:</i>	Two's complement representation of integers.	467
15.1.4	Representation of Real Numbers	468
<i>Fig. 15.4:</i>	Binary representation of reals.	468
15.1.5	Base Conversion	469
<i>Fig. 15.5:</i>	Converting binary numbers to decimal.	469
<i>Fig. 15.6:</i>	Converting between hexadecimal and binary.	469
<i>Fig. 15.7:</i>	Converting decimal numbers to binary.	469
<i>Fig. 15.8:</i>	Converting hexadecimal numbers to decimal.	469
15.2	Hexadecimal Input and Output	471
<i>Fig. 15.9:</i>	Hexadecimal character literals.	471
<i>Fig. 15.10:</i>	I/O for hex and decimal integers.	471
15.3	Bitwise Operators	472
<i>Fig. 15.11:</i>	Bitwise operators.	472
15.3.1	Masks and Masking	473
<i>Fig. 15.12:</i>	Bit masks for encrypting a number.	473
<i>Fig. 15.13:</i>	Truth tables for bitwise operators.	473
<i>Fig. 15.14:</i>	Bitwise AND (&).	474

<i>Fig. 15.15:</i>	Bitwise OR (<code> </code>).	474
<i>Fig. 15.16:</i>	Bitwise XOR (<code>^</code>).	474
<i>Fig. 15.17:</i>	Just say no.	474
15.3.2	Shift Operators	475
<i>Fig. 15.18:</i>	Left and right shifts.	475
15.3.3	Example: Shifting and Masking an Internet Address	476
<i>Fig. 15.19:</i>	Problem specifications: Decoding an Internet address.	476
<i>Fig. 15.20:</i>	Decoding an Internet address.	476
15.4	Application: Simple Encryption and Decryption	477
<i>Fig. 15.21:</i>	Problem specifications: Encrypting and decrypting a number.	478
<i>Fig. 15.22:</i>	Encrypting a number.	478
<i>Fig. 15.23:</i>	Decrypting a number.	478
15.5	Bitfield Types	481
<i>Fig. 15.24:</i>	Bitfield-structure demonstration.	482
15.5.1	Bitfield Application: A Device Controller (Advanced Topic)	484
<i>Fig. 15.25:</i>	Problem specifications: Skylight controller.	484
<i>Fig. 15.26:</i>	Declarations for the skylight controller: <code>skylight.hpp</code> .	486
<i>Fig. 15.27:</i>	A skylight controller.	486
<i>Fig. 15.28:</i>	Operations for the skylight controller.	488
15.6	What You Should Remember	491
15.6.1	Major Concepts	491
<i>Fig. 15.29:</i>	Summary of bitwise operators.	491
15.6.2	Programming Style	492
15.6.3	Sticky Points and Common Errors	492
15.6.4	New and Revisited Vocabulary	493
15.7	Exercises	493
15.7.1	Self-Test Exercises	493
15.7.2	Using Pencil and Paper	494
15.7.3	Using the Computer	495
<i>Fig. 15.30:</i>	Problem 10.	499

V Pointers 501

16 Dynamic Arrays 503

16.1	Operator Definitions	503
16.2	Pointers—Old and New Ideas	503
16.2.1	Pointer Declarations and Initialization	504
<i>Fig. 16.1:</i>	Pointing at an array.	504
16.2.2	Using Pointers	504
<i>Fig. 16.2:</i>	Indirect reference.	504
<i>Fig. 16.3:</i>	Direct and indirect assignment.	505
<i>Fig. 16.4:</i>	Input and output through pointers.	505
16.2.3	Pointer Arithmetic and Logic	506
<i>Fig. 16.5:</i>	Pointer addition and subtraction.	506
<i>Fig. 16.6:</i>	Incrementing a pointer.	507
<i>Fig. 16.7:</i>	Pointer comparisons.	507
16.2.4	Using Pointers with Arrays	507
<i>Fig. 16.8:</i>	An array with a cursor and a sentinel.	508
<i>Fig. 16.9:</i>	An increment loop.	508
16.3	Dynamic Memory Allocation	509
<i>Fig. 16.10:</i>	Allocating new memory.	510

16.3.1	Simple Memory Allocation	510
16.4	Arrays that Grow	512
16.4.1	The Flex Class	512
	Fig. 16.11: Flex: a growing array.	512
	Fig. 16.12: Flex functions.	512
16.5	Application: Insertion Sort	515
	Fig. 16.13: Inner loop of the Insertion sort.	515
	Fig. 16.14: Insertion sort.	515
	Fig. 16.15: Insertion sort using a dynamic array.	516
	Fig. 16.16: Insertion sort.	516
	Fig. 16.17: The Sorter class.	518
	Fig. 16.18: Insertion sort using a Flex array.	519
16.6	Selection Sort	521
	Fig. 16.19: Selection sort using a dynamic array.	521
	Fig. 16.20: The controller class: Charges.	521
	Fig. 16.21: Functions for the Charges class.	523
	Fig. 16.22: The data class: Transaction.	523
	Fig. 16.23: Functions for the data class, Transaction.	523
16.7	What You Should Remember	526
16.7.1	Major Concepts	526
16.7.2	Programming Style	527
16.7.3	Sticky Points and Common Errors	527
16.7.4	Vocabulary	528
16.7.5	Self-Test Exercises	529
16.7.6	Using Pencil and Paper	530
16.7.7	Using the Computer	531
	Fig. 16.24: Poker hands, high to low.	531
17	Vectors and Iterators	533
17.1	The Standard Template Library--STL	533
17.1.1	Containers	534
17.2	The vector Class	534
	Fig. 17.1: Vector operations.	534
	Fig. 17.2: Vector iterators.	535
17.2.1	Using the STL vector Class	535
	Fig. 17.3: Vector demo program.	535
	Fig. 17.4: Problem Specification.	537
17.3	A 3-D dynamic object.	538
	Fig. 17.5: The Pleasant Lakes Club.	538
	Fig. 17.6: The FamilyT class.	540
	Fig. 17.7: The FamilyT functions.	540
17.4	Using Vectors: A Simulation	543
17.4.1	Transient Heat Conduction in a Semi-Infinite Slab	543
	Fig. 17.8: Heat conduction in a semi-infinite slab.	543
17.4.2	Simulating the Cooling Process	544
	Fig. 17.9: A call chart for the heat flow simulation.	544
	Fig. 17.10: A heat flow simulation.	544
	Fig. 17.11: The Slab class.	544
	Fig. 17.12: The Slab constructor and destructor.	545
	Fig. 17.13: <code>simCool()</code> : doing the simulation.	547
	Fig. 17.14: <code>Print()</code> , <code>results()</code> , and <code>debug()</code>	548
17.5	What You Should Remember	550

17.5.1	Major Concepts	550
17.5.2	Programming Style	550
17.5.3	Sticky Points and Common Errors	551
17.5.4	New and Revisited Vocabulary	551
17.6	Exercises	551
17.6.1	Self-Test Exercises	551
17.6.2	Using Pencil and Paper	552
17.6.3	Using the Computer	552
18	Array Data Structures	555
18.1	Concepts	555
18.1.1	Declarations and Memory Layout	555
	Fig. 18.1: A two-dimensional array and initializer.	555
	Fig. 18.2: Layout of an array in memory.	555
18.1.2	Using <code>typedef</code> for Two-Dimensional Arrays	556
	Fig. 18.3: Using <code>typedef</code> for 2D arrays.	556
18.1.3	Ragged Arrays	556
	Fig. 18.4: A menu function.	556
18.1.4	A Matrix	558
	Fig. 18.5: Travel time for a two-day trip.	558
18.1.5	An Array of Arrays	560
	Fig. 18.6: Declaring an array of arrays.	561
	Fig. 18.7: Calculating wind speed.	561
	Fig. 18.8: Printing the wind speed table.	561
18.1.6	Dynamic Matrix: An Array of Pointers	563
18.1.7	Multidimensional Arrays	564
	Fig. 18.9: An array of dynamic arrays.	564
	Fig. 18.10: A three-dimensional array.	564
	Fig. 18.11: Problem specifications: 2D or 3D point transformation.	564
18.2	Application: Transformation of 2D Point Coordinates	564
	Fig. 18.12: 2D point transformation—main program.	566
	Fig. 18.13: Three Class Declarations	566
	Fig. 18.14: Functions for the Transform class.	568
	Fig. 18.15: Functions for the Drawing class.	570
18.3	Application: Image Processing	570
18.3.1	Digital images.	570
	Fig. 18.16: Problem specifications: Image smoothing.	572
18.3.2	Smoothing an image.	572
	Fig. 18.17: Image smoothing—main program.	572
	Fig. 18.18: Getting parameters for the smoothing.	573
	Fig. 18.19: Prevent accidental overwriting of existing file.	573
	Fig. 18.20: The Pgm class.	576
	Fig. 18.21: The Pgm constructors.	578
	Fig. 18.22: Parsing and printing the header.	580
	Fig. 18.23: Writing the Pgm file.	581
	Fig. 18.24: Smoothing.	581
	Fig. 18.25: Results of image smoothing program.	583
18.4	Application: Gaussian Elimination	583
18.4.1	An Implementation of Gauss's Algorithm	584
	Fig. 18.26: Solving a system of linear equations.	585
	Fig. 18.27: A call chart for Gaussian elimination.	585
	Fig. 18.28: The Equation class declarations.	586

<i>Fig. 18.29:</i>	The Gauss class declaration.	586
<i>Fig. 18.30:</i>	Constructing the model.	588
<i>Fig. 18.31:</i>	Output functions for the Gauss class.	589
<i>Fig. 18.32:</i>	Output for the Equation class.	590
<i>Fig. 18.33:</i>	The solve() function.	591
<i>Fig. 18.34:</i>	Finding the next pivot row.	591
<i>Fig. 18.35:</i>	Equation::scale().	593
<i>Fig. 18.36:</i>	Equation::wipe().	593
18.5	What You Should Remember	593
18.5.1	Major Concepts	593
18.5.2	Programming Style	594
18.5.3	Sticky Points and Common Errors	595
18.5.4	New and Revisited Vocabulary	595
18.6	Exercises	596
18.6.1	Self-Test Exercises	596
18.6.2	Using Pencil and Paper	598
18.6.3	Using the Computer	599

VI Developing Sophistication

605

19 Recursion

607

19.1	Storage Classes	607
19.1.1	Automatic Storage	607
19.1.2	Static Storage	608
19.1.3	External Storage (Advanced Topic)	608
19.2	The Run-Time Stack (Advanced Topic)	609
19.2.1	Stack Frames	609
19.2.2	Stack Diagrams and Program Traces	610
<i>Fig. 19.1:</i>	The run-time stack.	610
<i>Fig. 19.2:</i>	The run-time stack for the gas pressure program.	611
19.3	Iteration and Recursion	611
19.3.1	The Nature of Iteration	611
19.3.2	The Nature of Recursion	612
<i>Fig. 19.3:</i>	A recursive program to find a minimum.	612
19.3.3	Tail Recursion	612
19.4	A Simple Example of Recursion	613
19.5	A More Complex Example: Binary Search	615
<i>Fig. 19.4:</i>	Call chart for binary search.	615
<i>Fig. 19.5:</i>	The binary search main program.	616
<i>Fig. 19.6:</i>	The Table class declaration.	616
<i>Fig. 19.7:</i>	The Table constructor.	617
<i>Fig. 19.8:</i>	Ensuring that the data is sorted.	618
<i>Fig. 19.9:</i>	Searching for numbers.	618
<i>Fig. 19.10:</i>	The binary search function.	620
<i>Fig. 19.11:</i>	Diagrams of a Binary Search: Ready to begin.	622
<i>Fig. 19.12:</i>	Second active call.	622
<i>Fig. 19.13:</i>	Third active call.	622
<i>Fig. 19.14:</i>	Fourth active call.	623
<i>Fig. 19.15:</i>	Call graph for the quicksort program.	623
19.6	Quicksort	623
<i>Fig. 19.16:</i>	The main program for quicksort().	624

Fig. 19.17:	Tester generates data for <code>quicksort()</code>	624
Fig. 19.18:	Implementation for the Tester class.	625
Fig. 19.19:	The Quick class.	625
Fig. 19.20:	The <code>quicksort()</code> function.	627
Fig. 19.21:	<code>sortToCutoff()</code>	627
Fig. 19.22:	Setting the pivot.	628
Fig. 19.23:	A diagram of <code>setPivot()</code>	628
Fig. 19.24:	The <code>partition()</code> function, the heart of <code>quicksort()</code>	630
Fig. 19.25:	The first pass through partition.	631
Fig. 19.26:	The first recursive call.	631
Fig. 19.27:	Finishing the recursions.	631
Fig. 19.28:	Finish sorting using insertion sort.	631
Fig. 19.29:	Insertion sort, first part.	633
Fig. 19.30:	Insertion sort, second part.	633
19.6.1	Possible Improvements	635
19.7	What You Should Remember	635
19.7.1	Major Concepts	635
19.7.2	Programming Style	638
19.7.3	Sticky Points and Common Errors	638
19.7.4	New and Revisited Vocabulary	639
19.8	Exercises	639
19.8.1	Self-Test Exercises	639
19.8.2	Using Pencil and Paper	640
19.8.3	Using the Computer	641
20	Command Line Arguments	643
20.1	Command-Line Arguments	643
20.1.1	The Argument Vector	643
Fig. 20.1:	The argument vector.	643
20.1.2	Decoding Arguments	643
Fig. 20.2:	Using command-line arguments.	644
Fig. 20.3:	The Sorter class.	645
Fig. 20.4:	The Sorter Constructor.	645
Fig. 20.5:	Reading the data.	648
Fig. 20.6:	Printing the sorted data.	648
Fig. 20.7:	Sort in ascending or descending order.	649
20.2	Functions with Variable Numbers of Arguments	651
Fig. 20.8:	The <code>fatal()</code> function.	651
20.3	Modular Organization	652
20.3.1	File Management Issues with Modular Construction	652
20.3.2	Building a Multimodule Program	654
20.3.3	Using a makefile.	655
Fig. 20.9:	Steps in building a program.	656
20.3.4	A Unix makefile.	657
Fig. 20.10:	A Unix makefile for <code>game</code>	658
A	The ASCII Code	661
B	The Precedence of Operators in C	663
Fig. B.1:	The precedence of operators in C.	663

C	Keywords	665
C.1	Preprocessor Commands	665
C.2	Control Words	665
C.3	Types and Declarations	665
C.4	Additional C++ Reserved Words	665
C.5	An Alphabetical List of C and C++ Reserved Words	666
D	Advanced Aspects C Operators	667
D.1	Assignment Combination Operators	667
	<i>Fig. D.1:</i> Assignment combinations.	667
	<i>Fig. D.2:</i> A parse tree for assignment combinations.	667
D.2	More on Lazy Evaluation and Skipping	667
	<i>Fig. D.3:</i> Lazy evaluation.	667
	<i>Fig. D.4:</i> Skip the whole operand.	668
	<i>Fig. D.5:</i> And nothing but the operand.	670
D.2.1	Evaluation Order and Side-Effect Operators	670
D.3	The Conditional Operator	670
	<i>Fig. D.6:</i> A flowchart for the conditional operator.	671
	<i>Fig. D.7:</i> A tree for the conditional operator.	671
D.4	The Comma Operator	672
D.5	Summary	672
	<i>Fig. D.8:</i> Complications in use of side-effect operators.	673
E	Dynamic Allocation in C	675
	<i>Fig. E.1:</i> Dynamic memory allocation functions.	675
E.0.1	Mass Memory Allocation	675
E.0.2	Cleared Memory Allocation	677
E.0.3	Freeing Dynamic Memory	677
E.0.4	Resizing an Array	678
F	The Standard C Environment	681
F.1	Built-in Facilities	681
F.2	Standard Files of Constants	681
	<i>Fig. F.1:</i> Minimum values.	682
F.3	The Standard Libraries and <code>main()</code>	682
F.3.1	The Function <code>main()</code>	682
F.3.2	Characters and Conversions	683
F.3.3	Mathematics	683
F.3.4	Input and Output	685
F.3.5	Standard Library	687
F.3.6	Strings	688
F.3.7	Time and Date	690
F.3.8	Variable-Length Argument Lists	691
F.4	Libraries Not Explored	691