FREE DISK INCLUDED FREE DISK INCLUDED FREE DISK INCLUDED

C FOR ENGINEERS

2ND EDITION

Brian Bramer & Susan Bramer

ARNOLD

2-005-481-1

2-005-481-1

C for Engineers

Brian Bramer B.Tech., Ph.D., C.Eng., M.I.E.E., M.I.E.E.E.
Principal Lecturer in Computing Science
Department of Computing Science
De Montfort University
Leicester

Susan Bramer B.A.
Lecturer in Computing Science
Department of Computing Science
De Montfort University
Leicester

جامعة البليدة
المكتبة المركزية

ARNOLD

A member of the Hodder Headline Group
LONDON • SYDNEY • AUCKLAND
Copublished in North, Central and South America by
John Wiley & Sons, Inc., New York • Toronto

Contents

Preface viii

Sources of information x

1 Computer systems: an overview 1
- 1.1 The ANSI C standard 1
- 1.2 Computer hardware and software 2
- 1.3 Instruction and data storage 4
- 1.4 Low- and high-level languages 6

2 Program design, coding and testing 8
- 2.1 General requirements for software systems 8
- 2.2 System analysis 9
- 2.3 Programming paradigms 9
- 2.4 Program design - stepwise refinement 10
- 2.5 Testing 15

3 Outline structure of a C program 16
- 3.1 Program modules and objects 16
- 3.2 Program creation, compilation and execution 18
- 3.3 Editing, compiling and executing the program 20

4 Starting to program in C 21
- 4.1 Program to display a message on the display screen 21
- 4.2 Program to calculate the area of a rectangle 22
- 4.3 Types, constants and variables 22
- 4.4 Statements and expressions 25
- 4.5 Function calling, function parameters and the function result 26
- 4.6 The standard input/output streams *stdout*, *stderr* and *stdin* 30
- 4.7 Review of Program 4_3 30

5 Using standard library functions 31
- 5.1 Introduction to the *printf* function 31
- 5.2 Introduction to mathematical functions (sqrt, sin & cos) 32
- 5.3 Numeric keyboard input using the *scanf* function 35
- 5.4 The type qualifier *const* and *named constants* 37
- 5.5 The preprocessor #define directive and *symbolic constants* 38

6 Program layout, documentation and debugging 40
- 6.1 The C character set 40
- 6.2 Programming style: documentation and layout 40
- 6.3 Defining new type names using *typedef* 43
- 6.4 Detection of compile, link and run-time errors 43

7 Integral data types 48
- 7.1 Integral numeric data types 48
- 7.2 Integer number constants 49
- 7.3 Printing and reading integral types 50
- 7.4 The enumerative type 51

8 Real number types **53**
8.1 Real number constants 53
8.2 The accuracy of real number calculations 54
8.3 Conversion between real types 55
8.4 Printing and reading real numbers 55

9 Expressions and operators **59**
9.1 Type conversion in expressions 59
9.2 Operator *precedence* and *associativity* 61
9.3 Arithmetic operators 62
9.4 The *bitwise* operators 64
9.5 Casting or conversion operators 65
9.6 The *sizeof* operator 65

10 Assignment operators **68**
10.1 The = assignment operator 68
10.2 Type conversion across assignment 69
10.3 Increment and decrement operators 71
10.4 The compound assignment operators 74

11 The *if* statement **76**
11.1 The *if* statement 76
11.2 Relational operators 77
11.3 Logical operators 79
11.4 Testing *scanf* for successful conversion 79
11.5 Compound statements 80
11.6 Nested *if* statements 81
11.7 The conditional operator *?* 83
11.8 Using diagnostic aids in <assert.h> 84

12 The *while* and *do* statements **85**
12.1 The *while* statement 85
12.2 The *do* statement 88
12.3 The *null* or empty statement 91
12.4 Infinite loops 92

13 The *for* statement **93**
13.1 The *for* statement 93
13.2 Program to evaluate capital growth at a given interest rate 94
13.3 The sequence operator , (comma) 96
13.4 Program to evaluate the factorial of an integer number 97

14 Further control statements **101**
14.1 The *continue* statement 101
14.2 The *break* statement 101
14.3 The *goto* statement 102
14.4 The *switch* statement 102

15 More on input and output **106**
15.1 Validation of numeric data input 106
15.2 Plotting a graph on a text display screen 108
15.3 Plotting a graph on a graphics display screen 111

16 Functions **115**
16.1 Function *definition* 115
16.2 The *return* statement 116
16.3 A function to square a number 117
16.4 Function *prototypes* 119
16.5 Function to raise a *float* to an positive *integer* exponent 120
16.6 Function declarations in the original C 122
16.7 Passing parameters using *pass by value* 122
16.8 Passing parameters *by reference* 123
16.9 Function to swap the values of two variables 125
16.10 Function to maintain a running mean 126
16.11 Summary: *pass by reference* 128
16.12 Efficiency: parameter passing and function result 128
16.13 Using preprocessor macros to generate *inline functions* 129

17 Towards larger programs **133**
17.1 Information hiding and the scope of identifiers 133
17.2 Declarations and definitions 133
17.3 Local identifiers and local scope 134
17.4 External identifiers and file scope 134
17.5 *Automatic* and *static* storage classes 137
17.6 Evaluating Fibonacci numbers using *static* local variables 137
17.7 The type qualifiers *const* and *volatile* 139
17.8 Summary: identifier scope 140
17.9 Storage classes 140

18 Multi-file programs **142**
18.1 Modular programming and multi-file programs 142
18.2 External linkage and external variables and functions 143
18.3 Internal linkage and *static* external variables and functions 145
18.4 Constructing and using header files 146
18.5 Conditional selection 149
18.6 Random number generators 150
18.7 Simulation of the flow of patients through a doctor's surgery 153
18.8 Side effects 155
18.9 ADTs - abstract data types 156
18.10 Summary: external identifiers, internal and external linkage 157

19 One-dimensional arrays **158**
19.1 Defining and initialising arrays 158
19.2 Accessing the elements of an array 159
19.3 Program to calculate the average of a sequence of real numbers 160
19.4 Arrays as function parameters 162
19.5 Summary: arrays as function parameters 168
19.6 Using *#define* and *enum* to specify the array size 169
19.7 Using an array to implement a stack ADT 170
19.8 Implementing ADTs (abstract data types) 173
19.9 Simulation of the flow of patients through an outpatient clinic 174

20 Characters and strings **177**
20.1 The character type *char* 177
20.2 Printing and reading characters 179
20.3 Direct character I/O from the operating system 182
20.4 The standard library <ctype.h>: functions to test characters 183
20.5 String constants 184
20.6 Defining and manipulating arrays of characters 185
20.7 String input and output 186
20.8 Program using functions to process strings 188
20.9 Array processing functions in <string.h> 195

21 Pointers **197**
21.1 Variables and addresses 197
21.2 Defining and using pointers 198
21.3 Arrays and pointers 200
21.4 *const* qualified pointers 206
21.5 The *null pointer* 206
21.6 Pointers to pointers 207
21.7 The generic pointer type *void* * 208
21.8 Arrays of pointers 209
21.9 Passing parameters to function *main* 209
21.10 Pointers and the *sizeof* operator 211

22 Functions and pointers **212**
22.1 Pointers as function parameters 212
22.2 Functions that return a pointer as the function result 212
22.3 String processing functions using pointers 213
22.4 Pointers should only contain valid addresses 221
22.5 When is an actual parameter an array or a simple variable? 222

23 Input and output functions **223**
23.1 Standard input/output streams 223
23.2 Opening, closing, deleting and renaming files 224
23.3 Character and string input and output 225
23.4 The *fprintf*, *printf* and *sprintf* functions 227
23.5 The *fscanf*, *scanf* and *sscanf* functions 228
23.6 Binary (unformatted) input and output 228

24 Structures **230**
24.1 Declaring and initialising structures 230
24.2 Accessing the members of a structure 231
24.3 Pointers to structures 232
24.4 Structures as function parameters 232
24.5 Structures as members of structures 233
24.6 An ADT (abstract data type) to store student records 233
24.7 An ADT to store complex numbers 245
24.8 Unions 249
24.9 Bitfields 251

25 Dynamic storage allocation 255
25.1 Functions to allocate memory dynamically 255
25.2 Using dynamic storage allocation to implement a stack 257
25.3 Linked lists 262
26 Multi-dimensional arrays 266
26.1 Declaring and accessing multi-dimensional arrays 266
26.2 Multi-dimensional arrays as function parameters 267
26.3 Functions to read, write and multiply a two-dimensional matrix 268
26.4 Dynamically allocating multi-dimensional arrays 273
26.5 Two-dimensional arrays using an array of pointers to row data 280
26.6 Efficiency considerations when manipulating matrices 283
27 Evaluating mathematical series 284
27.1 Truncating infinite series 284
27.2 Program to evaluate the sine function sin(x) 285
27.3 Rounding errors 287
27.4 Evaluation of square root using Newton-Raphson method 291
28 Advanced use of functions 294
28.1 Recursive functions 294
28.2 Pointers to functions 295
28.3 Passing pointers to functions as function parameters 296
28.4 Returning pointers to functions as a function result 299
28.5 Functions with a variable number of parameters 302
29 Accessing operating system facilities 305
29.1 Hardware and software interrupts 305
29.2 Accessing MS-DOS operating system facilities 306
29.3 Using the RS232 serial port via MS-DOS 309
29.4 Using a Microsoft compatible mouse under MS-DOS 315
29.5 Invoking processes 317
30 Using assembly language from C 319
30.1 Separate assembly language functions 319
30.2 In-line assembly language statements 322
30.3 Interfacing other high-level languages with C 323
31 Low-level hardware programming 324
31.1 Setting up pointers to 'point to' locations in physical memory 324
31.2 Directly accessing video RAM of an IBM PC compatible computer 325
31.3 Accessing input/output device registers 328
31.4 Polled and interrupt I/O programming 328
31.5 Directly accessing the serial ports of an IBM PC compatible 328
31.6 Interrupt control of the serial ports of an IBM PC compatible 332
Appendix A The ASCII character code 342
Appendix B Answers to the exercises 343
Appendix C The standard library 380
Appendix D Table of C operators 394
References 395
Index 396