Contents at a Glance

Introduction ........................................................................................................... 1

Part I: Jumping Right In

1 What Is C Programming, and Why Should I Care? ........................................... 5
2 Writing Your First C Program ........................................................................ 13
3 What Does This Do? Clarifying Your Code with Comments ......................... 23
4 Your World Premiere—Putting Your Program’s Results Up on the Screen ........ 31
5 Adding Variables to Your Programs .................................................................. 41
6 Adding Words to Your Programs ...................................................................... 49
7 Making Your Programs More Powerful with #include and #define ............... 57
8 Interacting with Users ....................................................................................... 65

Part II: Putting C to Work for You with Operators and Expressions

9 Crunching the Numbers—Letting C Handle Math for You ....................... 73
10 Powering Up Your Variables with Assignments and Expressions ............... 83
11 The Fork in the Road—Testing Data to Pick a Path ..................................... 91
12 Juggling Several Choices with Logical Operators ..................................... 103
13 A Bigger Bag of Tricks—Some More Operators for Your Programs .......... 115

Part III: Fleshing Out Your Programs

14 Code Repeat—Using Loops to Save Time and Effort .................................. 123
15 Looking for Another Way to Create Loops ................................................ 131
16 Breaking in and out of Looped Code .......................................................... 141
17 Making the case for the switch Statement ................................................ 149
18 Increasing Your Program’s Output (and Input) ........................................... 163
19 Getting More from Your Strings ................................................................. 171
20 Advanced Math (for the Computer, Not You!) ............................................. 181

Part IV: Managing Data with Your C Programs

21 Dealing with Arrays ....................................................................................... 193
22 Searching Arrays .......................................................................................... 201
23 Alphabetizing and Arranging Your Data ..................................................... 209
24 Solving the Mystery of Pointers ................................................................... 221
25 Arrays and Pointers ...................................................................................... 231
26 Maximizing Your Computer’s Memory ....................................................... 243
27 Setting Up Your Data with Structures ......................................................... 257
Part V: Files and Functions

28  Saving Sequential Files to Your Computer ........................................... 267
29  Saving Random Files to Your Computer ............................................... 277
30  Organizing Your Programs with Functions ........................................... 285
31  Passing Variables to Your Functions ..................................................... 293
32  Returning Data from Your Functions ................................................... 305

Appendixes

A  The ASCII Table ..................................................................................... 313
B  The Draw Poker Program ..................................................................... 319

Index ............................................................................................................ 331
Table of Contents

Introduction ...................................................................................................................... 1
  Who's This Book For? ................................................................................................. 2
  What Makes This Book Different? ............................................................................ 2
  This Book's Design Elements .................................................................................. 3
  How Can I Have Fun with C? .................................................................................... 4
  What Do I Do Now? ................................................................................................... 4

Part I: Jumping Right In

1 What Is C Programming, and Why Should I Care? ........................................... 5
  What Is a Program? .................................................................................................. 6
  What You Need to Write C Programs ..................................................................... 7
  The Programming Process ....................................................................................... 10
  Using C .................................................................................................................... 11

2 Writing Your First C Program .............................................................................. 13
  A Down-and-Dirty Chunk of Code ....................................................................... 14
  The main() Function .............................................................................................. 16
  Kinds of Data ............................................................................................................ 17
    Characters and C .................................................................................................... 18
    Numbers in C ......................................................................................................... 19
  Wrapping Things Up with Another Example Program ........................................ 21

3 What Does This Do? Clarifying Your Code with Comments .......................... 23
  Commenting on Your Code ...................................................................................... 24
  Specifying Comments ............................................................................................. 25
  Whitespace ............................................................................................................... 27
  A Second Style for Your Comments ..................................................................... 28

4 Your World Premiere—Putting Your Program’s Results Up on the Screen ....... 31
  How to Use printf() ............................................................................................... 32
    The Format of printf() ......................................................................................... 32
  Printing Strings ....................................................................................................... 33
10 Powering Up Your Variables with Assignments and Expressions ............... 83
   Compound Assignment ........................................................................ 84
   Watch That Order! ........................................................................... 88
   Typecasting: Hollywood Could Take Lessons from C ........................................ 88

11 The Fork in the Road—Testing Data to Pick a Path ..................................... 91
   Testing Data .................................................................................. 92
   Using if ....................................................................................... 93
   Otherwise...: Using else ................................................................ 96

12 Juggling Several Choices with Logical Operators ..................................... 103
   Getting Logical ............................................................................. 104
   Avoiding the Negative ................................................................... 109
   The Order of Logical Operators ...................................................... 111

13 A Bigger Bag of Tricks—Some More Operators for Your Programs .......... 115
   Goodbye if...else; Hello, Conditional ................................................. 116
   The Small-Change Operators: ++ and -- .......................................... 119
   Sizing Up the Situation .................................................................. 121

Part III: Fleshing Out Your Programs

14 Code Repeat—Using Loops to Save Time and Effort .............................. 123
   while We Repeat ........................................................................... 124
   Using while ................................................................................. 125
   Using do...while ......................................................................... 127

15 Looking for Another Way to Create Loops .............................................. 131
   for Repeat’s Sake! ......................................................................... 132
   Working with for .......................................................................... 134

16 Breaking in and out of Looped Code ....................................................... 141
   Take a break ................................................................................ 142
   Let’s continue Working .................................................................. 145
24 Solving the Mystery of Pointers ................................................................. 221
   Memory Addresses .................................................................................. 222
   Defining Pointer Variables .................................................................... 222
   Using the Dereferencing * .................................................................... 225

25 Arrays and Pointers ............................................................................. 231
   Array Names Are Pointers ..................................................................... 232
   Getting Down in the List ........................................................................ 233
   Characters and Pointers ........................................................................ 234
   Be Careful with Lengths ........................................................................ 234
   Arrays of Pointers ................................................................................ 236

26 Maximizing Your Computer’s Memory .................................................. 243
   Thinking of the Heap ........................................................................... 244
   But Why Do I Need the Heap? ............................................................... 245
   How Do I Allocate the Heap? ............................................................... 246
   If There’s Not Enough Heap Memory .................................................. 249
   Freeing Heap Memory .......................................................................... 250
   Multiple Allocations ............................................................................. 250

27 Setting Up Your Data with Structures .................................................. 257
   Defining a Structure ............................................................................ 258
   Putting Data in Structure Variables ..................................................... 262

Part V: Files and Functions

28 Saving Sequential Files to Your Computer .......................................... 267
   Disk Files ............................................................................................. 268
   Opening a File ...................................................................................... 268
   Using Sequential Files ......................................................................... 270

29 Saving Random Files to Your Computer .............................................. 277
   Opening Random Files ......................................................................... 278
   Moving Around in a File ...................................................................... 279
30 Organizing Your Programs with Functions .................................................. 285
   Form Follows C Functions ........................................................................... 286
   Local or Global? ......................................................................................... 290
31 Passing Variables to Your Functions ............................................................ 293
   Passing Arguments ........................................................................................ 294
   Methods of Passing Arguments ..................................................................... 294
      Passing by Value ....................................................................................... 295
      Passing by Address ................................................................................... 297
32 Returning Data from Your Functions ............................................................. 305
   Returning Values .......................................................................................... 306
   The return Data Type .................................................................................. 309
   One Last Step: Prototype .............................................................................. 309
   Wrapping Things Up .................................................................................... 312

Appendixes

A The ASCII Table .......................................................................................... 313
B The Draw Poker Program .............................................................................. 319

Index ............................................................................................................... 331
About the Authors

Greg Perry is a speaker and writer in both the programming and applications sides of computing. He is known for bringing programming topics down to the beginner’s level. Perry has been a programmer and trainer for two decades. He received his first degree in computer science and then earned a Master’s degree in corporate finance. Besides writing, he consults and lectures across the country, including at the acclaimed Software Development programming conferences. Perry is the author of more than 75 other computer books. In his spare time, he gives lectures on traveling in Italy, his second favorite place to be.

Dean Miller is a writer and editor with more than 20 years of experience in both the publishing and licensed consumer product businesses. Over the years, he has created or helped shape a number of bestselling books and series, including Teach Yourself in 21 Days, Teach Yourself in 24 Hours, and the Unleashed series, all from Sams Publishing. He has written books on C programming and professional wrestling, and is still looking for a way to combine the two into one strange amalgam.
Dedication

To my wife and best friend, Fran Hatton, who’s always supported my dreams and was an incredible rock during the most challenging year of my professional career.

Acknowledgments

Greg: My thanks go to all my friends at Pearson. Most writers would refer to them as editors; to me, they are friends. I want all my readers to understand this: The people at Pearson care about you most of all. The things they do result from their concern for your knowledge and enjoyment.

On a more personal note, my beautiful bride, Jayne; my mother, Bettye Perry; and my friends, who wonder how I find the time to write, all deserve credit for supporting my need to write.

Dean: Thanks to Mark Taber for considering me for this project. I started my professional life in computer book publishing, and it is so gratifying to return after a 10-year hiatus. I’d like to thank Greg Perry for creating outstanding first and second editions upon which this version of the book is based. It was an honor working with him as his editor for the first two editions and a greater honor to coauthor this edition. I can only hope I did it justice. I appreciate the amazing work the editorial team of Mandie Frank, Krista Hansing, and the production team at Pearson put into this book.

On a personal level, I have to thank my three children, John, Alice, and Maggie and my wife Fran for their unending patience and support.
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn’t like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book and may not be able to reply personally to every message we receive.

When you write, please be sure to include this book’s title, edition number, and authors, as well as your name and contact information. We will carefully review your comments and share them with the authors and editors who worked on the book.

Email: feedback@quepublishing.com
Mail: Que Publishing
     800 East 96th Street
     Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at http://informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
Are you tired of seeing your friends get C programming jobs while you’re left out in the cold? Would you like to learn C but just don’t have the energy? Is your old, worn-out computer in need of a hot programming language to spice up its circuits? This book is just what the doctor ordered!

C Programming Absolute Beginner’s Guide breaks the commonality of computer books by talking to you at your level without talking down to you. This book is like your best friend sitting next to you teaching C. C Programming Absolute Beginner’s Guide attempts to express without impressing. It talks to you in plain language, not in “computerese.” The short chapters, line drawings, and occasionally humorous straight talk guide you through the maze of C programming faster, friendlier, and easier than any other book available today.
Who’s This Book For?

This is a beginner’s book. If you have never programmed, this book is for you. No knowledge of any programming concept is assumed. If you can’t even spell C, you can learn to program in C with this book.

The phrase absolute beginner has different meanings at different times. Maybe you’ve tried to learn C but gave up. Many books and classes make C much more technical than it is. You might have programmed in other languages but are a beginner in C. If so, read on, O faithful one, because in 32 quick chapters, you’ll know C.

What Makes This Book Different?

This book doesn’t cloud issues with internal technical stuff that beginners in C don’t need. We’re of the firm belief that introductory principles have to be taught well and slowly. After you tackle the basics, the “harder” parts never seem hard. This book teaches you the real C that you need to get started.

C can be an extremely cryptic and difficult language. Many people try to learn C more than once. The problem is simply this: Any subject, whether it be brain surgery, mail sorting, or C programming, is easy if it’s explained properly. Nobody can teach you anything because you have to teach yourself—but if the instructor, book, or video doing the teaching doesn’t make the subject simple and fun, you won’t want to learn the subject.

We challenge you to find a more straightforward approach to C than is offered in the C Programming Absolute Beginner’s Guide. If you can, call one of us because we’d like to read it. (You thought maybe we’d offer you your money back?) Seriously, we’ve tried to provide you with a different kind of help from that which you find in most other places.

The biggest advantage this book offers is that we really like to write C programs—and we like to teach C even more. We believe that you will learn to like C, too.
This Book’s Design Elements

Like many computer books, this book contains lots of helpful hints, tips, warnings, and so on. You will run across many notes and sidebars that bring these specific items to your attention.

**TIP** Many of this book’s tricks and tips (and there are lots of them) are highlighted as a Tip. When a really neat feature or code trick coincides with the topic you’re reading about, a Tip pinpoints what you can do to take advantage of the added bonus.

**NOTE** Throughout the C language, certain subjects provide a deeper level of understanding than others. A Note tells you about something you might not have thought about, such as a new use for the topic being discussed.

**WARNING** A Warning points out potential problems you could face with the particular topic being discussed. It indicates a warning you should heed or provides a way to fix a problem that can occur.

Each chapter ends by reviewing the key points you should remember from that chapter. One of the key features that ties everything together is the section titled “The Absolute Minimum.” This chapter summary states the chapter’s primary goal, lists a code example that highlights the concepts taught, and provides a code analysis that explains that code example. You’ll find these chapter summaries, which begin in Chapter 2, “Writing Your First C Program,” to be a welcome wrap-up of the chapter’s main points.

This book uses the following typographic conventions:

- Code lines, variables, and any text you see onscreen appears in monospace.
- Placeholders on format lines appear in *italic* monospace.
- Parts of program output that the user typed appear in **bold** monospace.
- New terms appear in *italic*.
- Optional parameters in syntax explanations are enclosed in flat brackets ([ ]). You do not type the brackets when you include these parameters.
How Can I Have Fun with C?

Appendix B, “The Draw Poker Program,” contains a complete, working Draw Poker program. The program was kept as short as possible without sacrificing readable code and game-playing functionality. The game also had to be kept generic to work on all C compilers. Therefore, you won’t find fancy graphics, but when you learn C, you’ll easily be able to access your compiler’s specific graphics, sound, and data-entry routines to improve the program.

The program uses as much of this book’s contents as possible. Almost every topic taught in this book appears in the Draw Poker game. Too many books offer nothing more than snippets of code. The Draw Poker game gives you the chance to see the “big picture.” As you progress through this book, you’ll understand more and more of the game.

What Do I Do Now?

Turn the page and learn the C language.
This page intentionally left blank
WRITING YOUR FIRST C PROGRAM

You get to see your first C program in this chapter! Please don’t try to understand every character of the C programs discussed here. Relax and just get familiar with the look and feel of C. After a while, you will begin to recognize elements common to all C programs.
A Down-and-Dirty Chunk of Code

This section shows you a short but complete C program and discusses another program that appears in Appendix B, “The Draw Poker Program.” Both programs contain common and different elements. The first program is extremely simple:

```c
/* Prints a message on the screen */
#include <stdio.h>
main()
{
    printf("Just one small step for coders. One giant leap for");
    printf(" programmers!\n");
    return 0;
}
```

Open your programming software and type in the program as listed. Simple, right? Probably not the first time you use your new compiler. When you open Code::Blocks for the first time, you will be greeted by a “Tip of the Day.” These tips will come in handy later, but right now you can just get rid of it by clicking Close.

To create your program, Click the File Menu and select New. Choose Empty File from the options that appear on the submenu. Now you’ve got a nice clean file to start writing your seven-line program.

After you type in your program, you will need to compile or build your program. To do this, click the little yellow gear icon in the upper-left corner. If you’ve typed the program in exactly and had no errors, you can then run the program by clicking the green right-facing arrow next to the gear. (The next icon in that row, with a gear and arrow, will do both the compiling and running of the program, simplifying your life by reducing the number of arduous clicks you must perform from two to one.)

When you compile (or build) the program and run it, you should see something like Figure 2.1.
FIGURE 2.1
The output of your first program.

NOTE Producing that one-line message took a lot of work! Actually, of the eight lines in the program, only two—the ones that start with `printf`—do the work that produces the output. The other lines provide “housekeeping chores” common to most C programs.

To see a much longer program, glance at Appendix B. Although the Draw Poker game there spans several pages, it contains elements common to the shorter program you just saw.

Look through both the programs just discussed and notice any similarities. One of the first things you might notice is the use of braces (`{ }`), parentheses (`()`), and backslashes (`\`). Be careful when typing C programs into your C compiler. C gets picky, for instance, if you accidentally type a square bracket (`[` when you should type a brace.

WARNING In addition to making sure you don’t type the wrong character, be careful when typing code in a word processor and then copying it to your IDE. I typed the previous program in Word (for this book) and then copied it to Code::Blocks. When compiling the program, I received a number of errors because my quotes on the `printf` line were smart quotes created by the word processor (to give that cool slanted look), and the compiler did not recognize them. After I deleted the quotes on the line and retyped them in my programming editor, the code compiled just fine. So if you get errors in programs, make sure the quotes are not the culprit.
C isn’t picky about everything. For instance, most of the spacing you see in C programs makes the programs clearer to people, not to C. As you program, add blank lines and indent sections of code that go together to help the appearance of the program and to make it easier for you to find what you are looking for.

**TIP** Use the Tab key to indent instead of typing a bunch of spaces. Most C editors let you adjust the tab spacing (the number of spaces that appear when you press Tab). Some C program lines get long, so a tab setting of three provides ample indentation without making lines too long.

C requires that you use lowercase letters for all commands and predefined functions. (You learn what a function is in the next section.) About the only time you use uppercase letters is on a line with `#define` and inside the printed messages you write.

# The main() Function

The most important part of a C program is its `main()` function. Both of the programs discussed earlier have `main()` functions. Although at this point the distinction is not critical, `main()` is a C function, not a C command. A function is nothing more than a routine that performs some task. Some functions come with C, and some are created by you. C programs are made up of one or more functions. Each program must always include a `main()` function. A function is distinguished from a command by the parentheses that follow the function name. These are functions:

```c
main() calcIt() printf() strlen()
```

and these are commands:

```c
return while int if float
```

When you read other C programming books, manuals, and webpages, the author might decide to omit the parenthesis from the end of function names. For example, you might read about the `printf` function instead of `printf()`. You’ll learn to recognize function names soon enough, so such differences won’t matter much to you. Most of the time, authors want to clarify the differences between functions and nonfunctions as much as possible, so you’ll usually see the parentheses.
WARNING One of the functions just listed, calcIt(), contains an uppercase letter. However, the preceding section said you should stay away from uppercase letters. If a name has multiple parts, as in doReportPrint(), it’s common practice to use uppercase letters to begin the separate words, to increase readability. (Spaces aren’t allowed in function names.) Stay away from typing words in all uppercase, but an uppercase letter for clarity once in a while is okay.

The required main() function and all of C’s supplied function names must contain lowercase letters. You can use uppercase for the functions that you write, but most C programmers stay with the lowercase function name convention.

Just as the home page is the beginning place to surf a website, main() is always the first place the computer begins when running your program. Even if main() is not the first function listed in your program, main() still determines the beginning of the program’s execution. Therefore, for readability, make main() the first function in every program you write. The programs in the next several chapters have only one function: main(). As you improve your C skills, you’ll learn why adding functions after main() improves your programming power even more. Chapter 30, “Organizing Your Programs with Functions,” covers writing your own functions.

After the word main(), you always see an opening brace ({). When you find a matching closing brace (}), main() is finished. You might see additional pairs of braces within a main() function as well. For practice, look again at the long program in Appendix B. main() is the first function with code, and several other functions follow, each with braces and code.

NOTE The statement #include <stdio.h> is needed in almost every C program. It helps with printing and getting data. For now, always put this statement somewhere before main(). You will understand why the #include is important in Chapter 7, “Making Your Programs More Powerful with #include and #define.”

Kinds of Data

Your C programs must use data made up of numbers, characters, and words; programs process that data into meaningful information. Although many different kinds of data exist, the following three data types are by far the most common used in C programming:
• Characters
• Integers
• Floating points (also called real numbers)

TIP You might be yelling “How much math am I going to have to learn?! I didn’t think that was part of the bargain!” Well, you can relax, because C does your math for you; you don’t have to be able to add 2 and 2 to write C programs. You do, however, have to understand data types so that you will know how to choose the correct type when your program needs it.

Characters and C

A C character is any single character that your computer can represent. Your computer knows 256 different characters. Each of them is found in something called the ASCII table, located in Appendix A, “The ASCII Table.” (ASCII is pronounced ask-ee. If you don’t know-ee, you can just ask-ee.) Anything your computer can represent can be a character. Any or all of the following can be considered characters:

A a 4 % Q ! + = ]

NOTE The American National Standards Institute (ANSI), which developed ANSI C, also developed the code for the ASCII chart.

TIP Even the spacebar produces a character. Just as C needs to keep track of the letters of the alphabet, the digits, and all the other characters, it has to keep track of any blank spaces your program needs.

As you can see, every letter, number, and space is a character to C. Sure, a 4 looks like a number, and it sometimes is, but it is also a character. If you indicate that a particular 4 is a character, you can’t do math with it. If you indicate that another 4 is to be a number, you can do math with that 4. The same holds for the special symbols. The plus sign (+) is a character, but the plus sign also performs addition. (There I go, bringing math back into the conversation!)

All of C’s character data is enclosed in apostrophes (‘). Some people call apostrophes single quotation marks. Apostrophes differentiate character data from other kinds of data, such as numbers and math symbols. For example, in a C program, all of the following are character data:
None of the following can be character data because they have no apostrophes around them:

A  a  4  %  -

TIP  None of the following are valid characters. Only single characters, not multiple characters, can go inside apostrophes.

‘C is fun’

‘C is hard’

‘I should be sailing!’

The first program in this chapter contains the character ‘\n’. At first, you might not think that \n is a single character, but it’s one of the few two-character combinations that C interprets as a single character. This will make more sense later.

If you need to specify more than one character (except for the special characters that you’ll learn, like the \n just described), enclose the characters in quotation marks ("). A group of multiple characters is called a string. The following is a C string:

“C is fun to learn.”

NOTE  That’s really all you need to know about characters and strings for now. In Chapters 4 through 6, you’ll learn how to use them in programs. When you see how to store characters in variables, you’ll see why the apostrophe and quotation marks are important.

Numbers in C

Although you might not have thought about it before now, numbers take on many different sizes and shapes. Your C program must have a way to store numbers, no matter what the numbers look like. You must store numbers in numeric variables. Before you look at variables, a review of the kinds of numbers will help.

Whole numbers are called integers. Integers have no decimal points. (Remember this rule: Like most reality shows, integers have no point whatsoever.) Any number without a decimal point is an integer. All of the following are integers:

10 54 0 -121 -68 752
WARNING  Never begin an integer with a leading 0 (unless the number is zero), or C will think you typed the number in hexadecimal or octal. Hexadecimal and octal, sometimes called base-16 and base-8, respectively, are weird ways of representing numbers. 053 is an octal number, and 0x45 is a hexadecimal number. If you don’t know what all that means, just remember for now that C puts a hex on you if you mess around with leading zeroes before integers.

Numbers with decimal points are called floating-point numbers. All of the following are floating-point numbers:

\[
\begin{align*}
547.43 & & 0.0 & & 0.44384 & & 9.1923 & & -168.470 & & .22
\end{align*}
\]

TIP  As you can see, leading zeroes are okay in front of floating-point numbers.

The choice of using integers or floating-point numbers depends on the data your programs are working with. Some values (such as ages and quantities) need only integers; other values (such as money amounts or weights) need the exact amounts floating-point numbers can provide. Internally, C stores integers differently than floating-point values. As you can see from Figure 2.2, a floating-point value usually takes twice as much memory as an integer. Therefore, if you can get away with using integers, do so—save floating points for values that need the decimal point.

---

**FIGURE 2.2**

*Storing floating-point values often takes more memory than integers.*
NOTE  Figure 2.2 shows you that integers generally take less memory than floating-point values, no matter how large or small the values stored there are. On any given day, a large post office box might get much less mail than a smaller one. The contents of the box don’t affect what the box is capable of holding. The size of C’s number storage is affected not by the value of the number, but by the type of the number.

Different C compilers use different amounts of storage for integers and floating-point values. As you will learn later, there are ways of finding out exactly how much memory your C compiler uses for each type of data.

Wrapping Things Up with Another Example Program

This chapter’s goal was to familiarize you with the “look and feel” of a C program, primarily the main() function that includes executable C statements. As you saw, C is a free-form language that isn’t picky about spacing. C is, however, picky about lowercase letters. C requires lowercase spellings of all its commands and functions, such as printf().

At this point, don’t worry about the specifics of the code you see in this chapter. The rest of the book explains all the details. But it is still a great idea to type and study as many programs as possible—practice will increase your coding confidence!

So here is a second program, one that uses the data types you just covered:

```c
/* A Program that Uses the Characters, Integers, and Floating-Point Data Types */
#include <stdio.h>

main()
{
    printf("I am learning the %c programming language\n", 'C');
    printf("I have just completed Chapter %d\n", 2);
    printf("I am %.1f percent ready to move on ", 99.9);
    printf("to the next chapter!\n");
    return 0;
}
```

This short program does nothing more than print three messages onscreen. Each message includes one of the three data types mentioned in this chapter: a character (C), an integer (2), and a floating-point number (99.9).
NOTE  On the first printf statement, the %c tells the program where to introduce the character 'C'. It is %c as an abbreviation for character, not because the character is a C. If you were learning the N programming language, you would still use %c to place the 'N' character.

The main() function is the only function in the program written by the programmer. The left and right braces ({} and {}) always enclose the main() code, as well as any other function's code that you might add to your programs. You’ll see another function, printf(), that is a built-in C function that produces output. Here is the program’s output:
I am learning the C programming language
I have just completed Chapter 2
I am 99.9 percent ready to move on to the next chapter!

TIP  Try playing around with the program, changing the messages or data. You should even try making a mistake when typing, like forgetting a semicolon (;) at the end of a line, just to see what happens when you try to compile the program. Learning from mistakes can make you a better programmer!

THE ABSOLUTE MINIMUM

This chapter familiarized you with the “look and feel” of a C program, primarily the main() function. The key points from this chapter include:

• A C function must have parentheses following its name. A C program consists of one or more functions. The main() function is always required. C executes main() before any other function.

• Put lots of extra spacing in your C programs, to make them more readable.

• Don’t put leading zeroes before integers unless the integer is zero.

• If you use a character, enclose it in single quotes. Strings go inside quotation marks. Integers are whole numbers without decimal points. Floating-point numbers have decimal points.
Symbols

#define directives, 60-62
#include directives, 58-60
-- operators, 119-121
++ operators, 119-121

A

addition operator, compound, 86
addPayroll() function, 292
addresses
    memory, 222
    passing arguments by, 297-302
allocating heap memory, 244-249
    multiple allocations, 250-255
American National Standards Institute (ANSI), 11, 18
ampersands, scanf() function, variables, 68-69
ANSI (American National Standards Institute), 11, 18
apostrophes (‘), character data, 18
arguments, 294
    passing, 293-294
        by address, 297-302
        by value, 295-297
arithmetic
    compound assignment operators, 84-87
        addition, 86
        multiplication, 86
        order, 88
    updating variables, 85-86
operators, 74-77
    assignment, 80-81
        order of, 77-79
    parentheses rules, 79
arrays, 52, 193, 231
    character, 52-54
        storing string literals, 234-236
    defining, 194-196
    elements, 53, 194-197
    filling, 202
    names, 232-233
    nums, 196
    parallel, 202
    passing, 303
    pointers, 236, 239-241
    putting values in, 197-199
    searching, 201-208
    sorting, 210
        ascending order, 210,
            214-215
        data searches, 215-220
        descending order, 210,
            214-215
    strings, printing in, 54
    subscripts, 196
    vals, 195
    ascending order, sorting arrays, 210, 214-215
    ASCll table, 313-317
    assignment operator, storing data in variables, 45
    assignment operators, 45,
        80-81
        compound, 84-87
            addition, 86
            multiplication, 86
            order, 88
            updating variables, 85-86
backslashes (/), 15
base-8 numbers, 20
base-16 numbers, 20
binary, 10
binary searches, arrays, 208
blocks, braces, 290
body, if statements, 94
braces ({), 15
    blocks, 290
break statement, 142-144,
    153-154
bubble sorting, arrays
    ascending order, 210,
        214-215
    data searches, 215-220
    descending order, 210,
        214-215
bugs, 10
buildContact() function, 288, 292

calcIt() function, 17
case, variables, checking,
    172-176
case statements, 153-162
case-changing functions, 176
c C compilers, 7
ceil() function, 182
char variables, 42
character arrays, 52-54
    storing string literals,
        234-236
character string literals, 43
character-testing functions, 172
characters, 18-19
    ASCII table, 313-317
    conversion, 36-37
keywords, extracting from, 164-167
pointers, 234
sending single character to screen, 164-167
strings, 19
closing files, 269
code
See also programming blocks, opening, 14
Blocks C compiler, 7-9
comments, 23-25
alternate style, 28
multi-line, 25
single-line, 28
specifying, 25-27
debugging, 10
indention, 16
line breaks, 27-28
loops
continue statement, 145-146
do...while, 127-129
for, 132-139
nesting, 135
terminating with break statement, 142-144
while, 124-129
output, printf() function, 31-39
source, 10
whitespace, 27-28
word processors, copying from, 15
commands, 11
do...while, repeating code, 127-129
while, 124
repeating code, 124-129
comments, 23-25
alternate style, 28
multi-line, 25
single-line, 28
specifying, 25-27
compilers, 7, 10
Blocks C compiler, 7-9
compound assignment operators, 84-87
addition, 86
multiplication, 86
order, 88
updating variables, 85-86
compound relational operators. See logical operators
compound statements, 94
counter variables, 84
cross-platform software, 7
data
literal, 42
saving, 267
sorting, 209
storing in variables, 45-48
structures, 257
defining, 258-262
putting data in, 262-265
testing
else statement, 96-100
if statement, 92-96
data searches, sorting arrays, 215-220
data types, 17-18
character, 18-19
floating-point numbers, 20-21
int, 258
integers, 19-20
mixing, 89
return, 309
variables, 42
deallocating heap memory, 244-246
debugging, 10
declaring
structures, 259
variables, 44-45
decrement operators, 74, 119-121
deficiencies, heap memory, 249
defined constants, 60-64
naming, 61
defining
arrays, 194-196
constants, #define directive, 60-62
pointer variables, 222-224
structures, 258-262
variables, 44-45, 60
same line, 44
dereferencing pointer variables, 225, 228
descending order, sorting arrays, 210, 214-215
disk files, 268-270
dot operator, 262
double variables, 42
do...while loops
repeating code, 127-129
terminating, 142-144
Draw Poker program, 14
comments, 25
functions, 289
header files, 60
main() function, 96
ELEMENTS

E

editors, 10
elements, arrays, 53, 194-197
else statement, testing data, 96-100
Enter keypress, terminating,
getchar() function, 167-168
equals sign, storing data in variables, 45
escape sequences, 34-36
printf() function, 34
exit() function, 153
expressions, 6, 74

F

fabs() function, 183-184
fclose() function, 269, 278
eof() function, 274
fgetc() function, 281
fgets() function, 235-236, 272
Fibonacci searches, arrays, 208
file pointers, 268
global, 269
files
closing, 269
disk, 268
header, 59
building, 62-64
Draw Poker program, 60
quotation marks, 59
including, #include preprocessor directives, 58-60
navigating, 279-284
opening, 268-270
pointer, 268
random-access, 268, 277-278
opening, 278-279
sequential, 268-275
filling arrays, 202
flag variables, 206
float variables, 42
floating-point absolute values, 183
floating-point numbers, 20-21
conversion characters, 36-37
floor() function, 182
fopen() function, 268-270, 278-279
for loops, 131-135, 138-139
nested, 210
relational test, 134
semicolons, 133
terminating, break statement, 142-144
formats, printf() function, 32-33
found variable, 206
fprintf() function, 270
fputc() function, 281
free() function, 246, 252
freeing heap memory, 250
fscanf() function, 274
fseek() function, 279-284
functions, 286-289
addPayroll(), 292
buildContact(), 288, 292
calct(), 17
case-changing, 176
ceil(), 182
c saldırı, 172
changing, 176
case-changing, 176
ceil(), 182
c săndar, 172
cancial, 164-169, 172
gets(), 177, 194, 235, 307
gradeAve(), 307-308
half(), 295-296
isalpha(), 172
isdecimal(), 172
islower(), 172-176
isupper(), 172-176
main(), 16-17, 21-22, 59-62,
96, 260, 285, 288, 295-296,
308-312
malloc(), 246-252
math, 181-184
generating random values,
187-188, 191
logarithmic, 184-186
trigonometric, 184-186
passing arguments, 293-294
by address, 297-302
by value, 295-297
pow(), 183
prAgain(), 291
printContact(), 288
printf(), 16, 22, 32, 49, 56, 59,
65-66, 118, 126, 195, 233,
270, 310
code output, 31-39
ccontrolString, 32-33
conversion characters,
36-37
escape sequences, 34-36
format, 32-33
placeholders, 32
printing strings, 33
prompting users before
scanf(), 66-68
prototypes, 305, 309-311
putc(), 281
putchar(), 164-167
puts(), 177, 195
rand(), 187-188, 191, 214
returning values, 306-309
scanf(), 65, 300
ampersands, 68-69
header file, 66
problems with, 68-71
prompting users with
printf(), 66-68
sizeof(), 196, 247
sqrt(), 183, 306
srand(), 187
strcpy(), 54, 59, 176-179, 194,
197, 234
string, 176-179
terminating, break statement, 142-144
while, 124-129

G
getchar() function, 164-169, 172
terminating Enter keypress, 167-168
getch() function, 172
gets() function, 177, 194, 235, 307
global file pointers, 269
local variables, 45, 290-292, 312
gradeAve() function, 307-308

H
half() function, 295-296
header files
building, 62-64
Draw Poker program, 60
quotation marks, 59
scanf() function, 66
heap memory, 243-246
allocating, 244-249
deallocating, 244-246
deficiencies, 249
freeing, 250
multiple allocations, 250-255
pointer variables, 243-244
hexadecimal numbers, 20

I-J
IDE (integrated development environment), 7
if...else statements, 96,
116-118, 150
if statement, 91, 149
body, 94
testing data, 92-96
increment operators, 119-121
local variables, 45, 290-292
logarithmic functions, 184-186
logical operators, 103-108
avoiding negative, 109-111
combining with relational
operators, 104-108
order, 111-112
loops, 123, 131
continue statement, 145-146
do...while, 127-129
for, 131-135, 138-139
nested, 210
relational test, 134
semicolons, 133
infinite, 123
nesting, 135

K-L
keywords, extracting single character from, getchar() function, 164-167
leading 0, integers, 20
leading spaces, control string characters, scanf() statements, 68
length, strings, 51-52
line breaks, 27-28
literal data, 42
memory, heap, 243-246
allocating, 244-249
deallocating, 244-246
deficiencies, 249
freeing, 250
multiple allocations, 250-255
pointer variables, 243-244

M
machine language, 10
main() function, 16-17, 21-22,
59, 62, 96, 260, 285, 288,
295-296, 308-312
#include directives, 60
maintenance, programs, 24
malloc() function, 246-252
math
compound assignment operators, 84-87
addition, 86
multiplication, 86
order, 88
updating variables, 85-86
operators, 74-77
assignment, 80-81
order of, 77-79
parentheses rules, 79
math functions, 181-184
generating random values, 187-191
logarithmic, 184-186
trigonometric, 184-186
members, 257
memory, heap, 243-246
allocating, 244-249
deallocating, 244-246
deficiencies, 249
freeing, 250
multiple allocations, 250-255
pointer variables, 243-244
memory addresses, 222
mixing data types, 89
mode strings, fopen() function, 270
modulus operator, 76
multi-line comments, 25
multiple allocations, heap memory, 250-255
multiplication operator, compound, 86

N

named constants, 60
naming
  defined constants, 61
  variables, 43-44
navigating files, 279-284
nested for loops, 210
nesting loops, 135
nonintegers variables, passing, 303
nonintegers, promoting/demoting, 182
null zeros, 50
numbers
  floating-point, 20-21
  hexadecimal, 20
  integers, 19-20
  octal, 20
nums array, 196

O

octal numbers, 20
open source software, 7
opening
  files, 268-270
  random-access files, 278-279
operators, 73-77
  assignment, 80-81
  variables, 45
  compound assignment, 84-87
  addition, 86
  multiplication, 86
  order, 88
  updating variables, 85-86
conditional, 116-118
decrement, 74, 119-121
dot, 262
increment, 119-121

logical, 103-108
  avoiding negative, 109-111
  combining with relational operators, 104-108
  order, 111-112
  modulus, 76
  order of, 77-79
  parentheses rules, 79
  postfix, 119
  prefix, 119
  relational, 91-92, 96, 103-104
  combining with logical operators, 104-108
  sizeof(), 121-122
order
  arrays, 210, 214-215
  compound assignment operators, 88
  logical operators, 111-112
  operators, 77-79
organizing programs, 285-289
origin values, fseek() function, 279
output, 7
  code, printf() function, 31-39
  programs, 14

P

parallel arrays, 202
parameters, 294
parentheses (()), 15
  logical operators, 111
  rules, operators, 79
passing
  arguments, 293-294
    by address, 297-302
    by value, 295-297
  arrays and nonarray variables, 303
placeholders, 32
placing #include directives, 60
pointers, 221, 231
  array names, 232-233
  arrays of, 236, 239-241
characters, 234
  constants, 232
  defining, 222-224
dereferencing, 225, 228
files, 268
  global, 269
heap memory, 243-244
memory addresses, 222
structure, 262
postfix operators, 119
pow() function, 183
prAgain() function, 291
prefix operators, 119
preprocessor directives, 57
  #define, 60-62
  #include, 58-60
    placing, 60
printContact() function, 288
  code output, 31-39
  controlString, 32-33
  conversion characters, 36-37
  escape sequences, 34-36
  format, 32-33
  placeholders, 32
  printing strings, 33
  prompting users before scanf(), 66-68
printing
  strings, 33
  strings in arrays, 54
programmers, 6
programming
  See also code
  process, 10
  requirements, 7-10
programs, 6-7
  building, 62-64
  Draw Poker, 14
    comments, 25
    functions, 289
    header files, 60
IDE (integrated development environment), 7
maintenance, 24
organizing, 285-289
output, 7, 14
writing, requirements, 7-10
prototypes (functions), 305, 309-311
putc() function, 281
putchar() function, 164-167
puts() function, 177, 195

Q-R

quotation marks (, characters, 19
header files, 59
rand() function, 187-188, 191, 214
random-access files, 268, 277-278
navigating, 279-284
opening, 278-279
random values, generating, 187-191
real numbers, 20-21
conversion characters, 36-37
records, 258
relational operators, 91-92, 96, 103-104
combining with logical operators, 104-108
relational tests, for loops, 134
return data type, 309
returning values, functions, 306-309

S

saving data, 267
scanf() function, 65, 300
header file, 66
problems with, 68-71
prompting users with printf(), 66-68
variables, ampersands, 68-69
searching arrays, 201-208
self-prototyping functions, 310

semicolons
commands and functions, 33
for loops, 133
sequential files, 268-275
closing, 269
opening, 268-270
sequential searches, arrays, 208
single-line comments, 28
sizeof() function, 121-122, 196, 247
software, cross-platform and open source, 7
sorting arrays, 209
ascending order, 210, 214-215
data searches, 215-216, 219-220
descending order, 210, 214-215
source code, 10
spacebar character, 18
spaces, control string characters, scanf() statements, 68
specifying comments, 25-27
sqrt() function, 183, 306
rand() function, 187
statements
break, 142-144, 153-154
case, 153-162
compound, 94
continue, 145-146
do...while, repeating code, 127-129
for, repeating code, 132-139
if, 91, 149
body, 94
testing data, 92-96
if...else, 96, 116-118, 150
invStruct, 260-262
struct, 258-259
switch, 150-154
while, repeating code, 124-129
storing data in variables, 45-48
equals sign, 45

strcpy() function, 54, 59, 176-179, 194, 197, 234
string functions, 176-179
string.h header file, 176
string literals, character arrays, 234-236
string terminator, 50
string variables, 49
strings, 19, 171
character arrays, 52-54
concatenation, 176
control, printf() function, 32
initializing, 54-56
length, 51-52
mode, fopen(), 270
printing, 33
printing in arrays, 54
string terminator, 50
terminating zero, 50-51
strlen() function, 176-179
struct statement, 258-259
structures, 257-258
declaring, 259
defining, 258-262
putting data in structure variables, 262-265
subscripts, 53
arrays, 196
switch statement, 150-154
syntax, code comments, 25-27

T

terminating loops, break statement, 142-144
terminating zero, strings, 50-51
testing data
else statement, 96-100
if statement, 92-96
tolower() function, 176
toupper() function, 129, 176, 240
trigonometric functions, 184-186
typecasting, 88-89
U-V

updating variables, compound assignment operators, 85-86
uppercase letters, defined constant names, 61

vals arrays, 195
values
  arrays, putting in, 197-199
  passing arguments by, 295-297
  returning, functions, 306-309
variables, 41-43, 294
  char, 42
  checking case, 172-176
  counter, 84
  data types, 42
  decrementing, 119
  defining, 44-45, 60
  double, 42
  flag, 206
  float, 42
  found, 206
  global, 45, 290-292, 312
  incrementing, 119
  incrementing counter, 132
  int, 42
  local, 45, 290-292
  naming, 43-44
  nonarray, passing, 303
  passing, 293-294
    by address, 297-302
    by value, 295-297
  pointers, 221, 231
    array names, 232-233
    arrays of, 236, 239-241
    characters, 234
    constants, 232
    defining, 222-224
    dereferencing, 225, 228
    heap memory, 243-244
    memory addresses, 222
  scanf() function, ampersands, 68-69
  storing data in, 45-48
  string, 49

structure, putting data in, 262-265
typecasting, 89
updating, compound assignment operators, 85-86
void keyword, 309

W-Z

while command, 124
while loops
  repeating code, 124-129
  terminating, 142-144
whitespace, 27-28
word processors, copying code from, 15
writing programs, requirements, 7-10

zeroes, terminating, strings, 50-51