C Programming

Updated for C11

ABSOLUTE BEGINNER'S GUIDE

No experience necessary!

Third Edition

que

Greg Perry and Dean Miller

C Programming

Greg Perry and Dean Miller

que®
800 East 96th Street
Indianapolis, Indiana 46240

C Programming Absolute Beginner's Guide
Third Edition

ISBN-13: 978-0-7897-5198-0
ISBN-10: 0-7897-5198-4

Library of Congress Control Number: 2013943628

Printed in the United States of America

Third Printing: January 2015

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

Bulk Sales

Que Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact

International Sales
international@pearsoned.com

Acquisitions Editor
Mark Taber

Managing Editor
Sandra Schroeder

Project Editor
Mandie Frank

Copy Editor
Krista Hansing Editorial Services, Inc.

Indexer
Brad Herriman

Proofreader
Anne Goebel

Technical Editor
Greg Perry

Publishing Coordinator
Vanessa Evans

Interior Designer
Anne Jones

Cover Designer
Matt Coleman

Compositor
TnT Design, Inc.

Contents at a Glance

Table of Contents

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

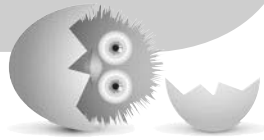
Visit our website and register this book at http://informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

IN THIS INTRODUCTION

- Who's This Book For?
- What Makes This Book Different?
- This Book's Design Elements
- How Can I Have Fun with C?
- What Do I Do Now?

INTRODUCTION



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?

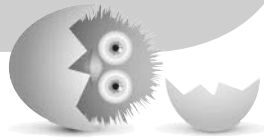
Turn the page and learn the C language.

This page intentionally left blank

IN THIS CHAPTER

- Typing your first program
- Using the `main()` function
- Identifying kinds of data

2



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:

```
/* 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.

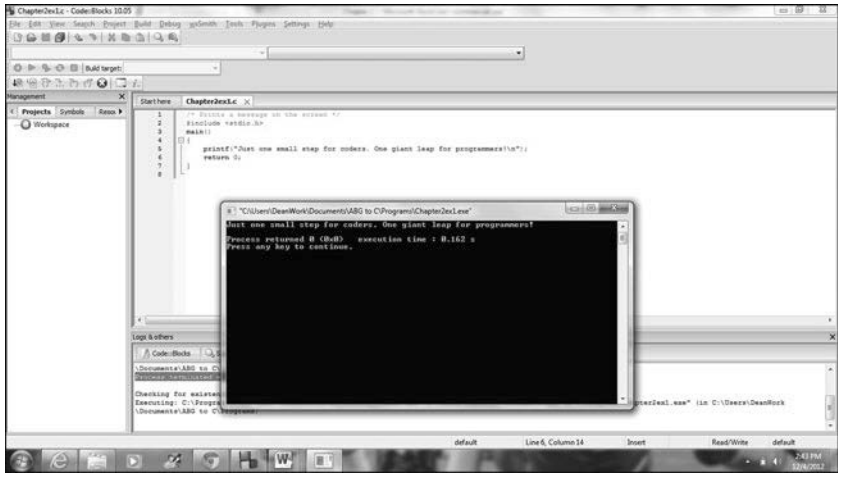


FIGURE 2.1

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 indention 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:

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

and these are commands:

```
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:

'A' 'a' '4' '%' ' ' '-'

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:

547.43 0.0 0.44384 9.1923 –168.470 .22

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.

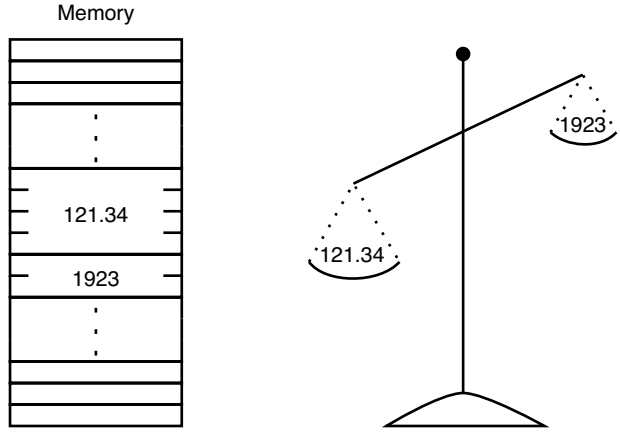


FIGURE 2.2

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:

```
/* 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:

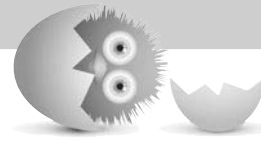
```
I am learning the C programming language
I have just completed Chapter 2
I am 99.9 percent ready to move on to the next chapter!
```

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.



Index

Symbols

A

B

C

D

E

F

N

O

P

Q-R

S

T

U-V

W-Z