IN THIS CHAPTER

- Reviewing software requirements
- Programming at the command prompt
- Using an IDE
- Creating a command prompt program
- Working in Code::Blocks
- Compiling a program

Chapter 1

A Quick Start for the Impatient

You're most likely eager to get started programming in C. I shan't waste your time.

TIP

If you already have a compiler or an IDE set up and are ready to go, skip to Chapter 2.

What You Need to Program

The two most important things you need to begin your programming adventure are

- A computer
- Access to the Internet

The computer is your primary tool for writing and compiling code. Yes, even if you're writing a game for the Xbox, you need a computer to be able to code. The computer can be a PC or a Macintosh. The PC can run Windows or Linux.

Internet access is necessary to obtain the programming software. You need a text editor to write the code and a compiler to translate the code into a program. The compiler generally comes with other tools you need, such as a linker and a debugger. All these tools are found at no cost on the Internet.

The software tools offer two approaches to programming: command line and IDE.

If you want to learn C programming as I did back in the dark ages, you use a terminal window and traditional command-line tools: editor, compiler, and linker. The process is fast, but complicated because you're using text mode commands. Still, it offers a spiritual connection with those who built the foundations upon which the computer industry roosts.

The most common way to craft code, however, is to obtain an integrated development environment — called an *IDE* by the cool kids. It combines all the tools you need for programming into one compact, terrifying, and intimidating unit.

Don't freak! The terms *compiler, linker, debugger*, and *terrifying* are all defined in Chapter 2.

Command Prompt Programming

To re-create the environment where the C language was born, use a Unix or Linux terminal window running a shell program such as *bash*. This environment is available to all major computing platforms, and the programming tools used are reliable and well-documented. Programming at the command prompt earns you a nerd merit badge and the admiration of your peers.

For Windows 10, open the Microsoft Store app and install Ubuntu, a free Linux shell. Ensure that you follow the directions to install the Windows Subsystem for Linux, which is an extra step you'll probably miss.

For Linux, you're ready to go: Open a terminal window to access the shell.

For Mac OS X, use the Terminal app. I also recommend obtaining the Homebrew package manager. Visit `https://brew.sh` for directions. Homebrew allows you to install programming tools not available to OS X.

For an editor, you can use any text mode editor available at the command prompt, such as *vi* or *emacs*. You can also "mix it up" and use a window-based editor. I'm fond of using the Windows version of the VIM editor while I simultaneously work at the command prompt in an Ubuntu terminal window.

A C compiler comes native to a Unix/Linux command prompt. The standard version is *cc* or *gcc*, but I recommend that you use the shell's package manager to acquire the LLVM *clang* compiler. In Ubuntu Linux for Windows 10, type this command to install *clang:*

```
sudo apt-get install clang
```

Type your account password to initiate the process. To verify the installation, type

```
clang --version
```

Various Linux distros offer similar package managers, which you can use to obtain an editor and the *clang* compiler.


REMEMBER

- The VIM editor can be obtained from `vim.org`.
- Your choice to use the command prompt means you're taking on an extra layer of complexity when it comes to programming. I find it fast and enjoyable, but if you believe it to be too much, especially when first learning the C language, rely instead upon an IDE, as covered in the next section.

IDE Programming

Plenty of programming IDEs are available for your C coding pleasure. On the Mac, use Xcode, which you can install from the App Store. For Windows and Linux, I recommend obtaining the Code::Blocks IDE, which is found at `codeblocks.org`. You can choose any other IDE you prefer, but Code::Blocks for Windows is fairly stable and comes with everything you need — providing that you install the correct version.

Installing Code::Blocks

The Code::Blocks website will doubtless be altered over time, so follow these general steps to install the IDE and confirm that the C compiler is accessible:

1. **On the main Code::Blocks website page, click the Downloads link.**
2. **Click the binary release link.**

   The "binary release" means you're installing a runnable program, not source code or something equally strange.
3. **Choose the proper installation program for your computer's operating system.**

   For Windows 10, I recommend that you choose the installation with the text *mingw-setup* appended. For example:

   ```
   codeblocks-20.03mingw-setup.exe
   ```

   The `20.03` part of the filename is the release number, which will change in the future. The `mingw-setup` choice means you're downloading both the IDE and the MinGW compiler.


TIP

   For Linux, click the link to install the proper version for your distro, but keep in mind that Code::Blocks might be more easily acquired by using the Linux GUI package/software manager.
4. **Open the downloaded archive to extract the Code::Blocks IDE installer.**

   In Windows, you see a User Account Control warning when you open the archive. Click Yes to proceed with installation.
5. **Run the installation program.**

   Perform a default installation; you need not customize anything.
6. **Choose to run Code:Blocks: Click the Yes button.**

   Code::Blocks appears, showing its splash screen. Don't start coding now. Instead, confirm the compiler's installation:
7. **Choose Settings, Compiler.**

   The Compiler Settings dialog box appears.
8. **With Global Compiler Settings chosen on the left, click the Toolchain Executables tab on the right side of the dialog box.**
9. **Ensure that the Compiler's Installation Directory text box is filled.**

   In a default confirmation, the following pathname is listed:

   ```
   C:\Program Files (x86)\CodeBlocks\MinGW
   ```

If the text box is blank, use the Browse button (three dots to the right of the text box) to locate the `MinGW` installation directory.

10. **Confirm that** gcc.exe **is set in the Compiler text box.**

    If not, click the Browse button (three dots) to locate the `gcc.exe` program, installed in the `MinGW\bin` directory by default.

11. **Close the Compiler Settings dialog box; click OK.**

Installation is complete. I recommend you close the Code::Blocks window. Finish the installation program as well.

Touring the Code::Blocks workspace

Figure 1-1 illustrates the Code::Blocks *workspace*, which is the official name of the massive mosaic of windows and toolbars you see arranged on the screen.

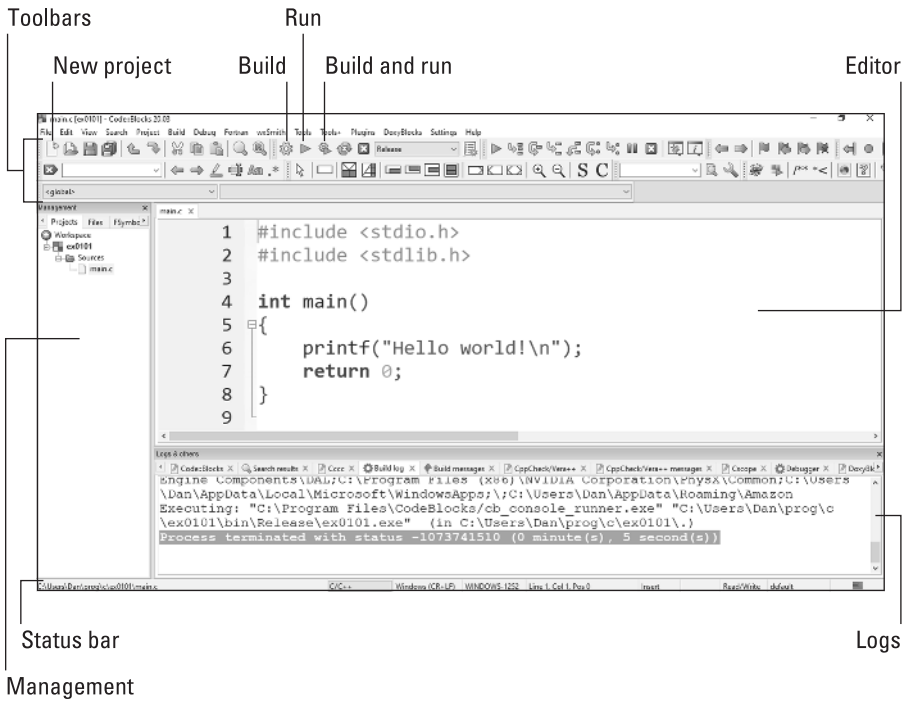

FIGURE 1-1: The Code::Blocks workspace.

On your computer, as well as in Figure 1-1, locate the following parts of the workspace:

Toolbars: These messy strips, adorned with various command buttons, cling to the top of the Code::Blocks window. The toolbars can be rearranged or hidden, so don't mess with them until you get comfy with the interface.

Management: The pane on the left side of the workspace features four tabs, though you may not see all four at one time. The window provides a handy oversight of your programming endeavors.

Status bar: At the bottom of the screen, you see information about the project, editor, and other activities that take place in Code::Blocks.

Editor: The big window in the center-right area of the screen is where you type code.

Logs: The bottom of the screen features a window with many, many tabs displaying various tidbits about the programming process. The tab used most often is named Build Log.

The View menu controls the visibility of every item displayed in the window. Choose the pane name, such as Manager, from the View menu to show or hide that item. Control toolbars by using the View, Toolbars submenu.

TIP

- Maximize the Code::Blocks program window so that it fills the screen. You need all that real estate.
- In addition to color-coding your text, the Code::Blocks editor offers an autocomplete feature. Items that must be typed in pairs, such as quotes, parentheses, and so on, are generated automatically for you. Certain C language keywords and functions are presented automatically, along with hints for their arguments and options.
- The editor features a tabbed interface, which lets you work on multiple source code files at one time.

Your First Program

The traditional first program written for any programming language is called Hello World. It's boring, like all traditions.

Listing 1-1 shows the Hello World source code as presented in Code::Blocks. This code is generated by default whenever you start a new Code::Blocks project.

LISTING 1-1: **The Code::Blocks Skeleton**

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

The programming process works the same whether you use the command prompt or an IDE:

1. **Use an editor to write the source code.**
2. **Compile and link the source code into a program.**
3. **Run the program to see whether it works.**

Chapter 2 goes over these steps in detail, but I assume you're in a rush. The following sections cover the specifics for the command prompt and IDE environments.

Coding at the command prompt

Use your text editor to create a new file and type in the text presented in Listing 1-1. This code, like all source code in this book, is available on the companion website: `https://c-for-dummies.com/cprog`.

Save the source code file as `ex0101.c`. The filename must end in `.c` ("dot C") to be recognized as a C source code file by the compiler.

Compile and link, or "build," the source code into a program file in a Unix terminal window, such as the Ubuntu bash shell in Windows 10. Type the following command in the same folder/directory where the source code file is saved:

```
clang -Wall ex0101.c
```

The name of the compiler program is *clang*. The `-Wall` argument activates all warning messages. The final argument is the name of the source code file, `ex0101.c`, in this example.

Upon success, you see no output. If you see warnings or error messages, you probably mistyped the source code. Try again: re-edit and compile.

To run the program, type the default program filename `a.out`. In a terminal window, the program name must be prefixed by `./` ("dot slash") to direct the command interpreter to look in the current directory:

```
./a.out
```

You see the message `Hello, world!` output in the terminal window. Now skim to Chapter 2 to discover what just happened.

Building a new Code::Blocks project

Two approaches can be used in Code::Blocks to build the sample code shown in this book: You can build a project for each exercise or you can use an empty file to write the code.

Create a project

For a project, follow these steps:

1. **Choose File, New, Project.**
2. **Choose Console Application and then click the Go button.**
3. **Select C as the programming language and then click the Next button.**

   C is quite different from C++ — you can do things in one language that aren't allowed in the other.
4. **Type a project title.**

   The code exercises in this book follow a naming pattern you can use for the project title: **ex** followed by a 2-digit chapter number and 2-digit sequential number. For the sample code presented in Listing 1-1, the project title is **ex0101**.

   When you set the project title, the project's filename is automatically filled in.
5. **Click the . . . (Browse) button to the right of the text box titled Folder to Create Project In.**

   I recommend that you create and use a special folder for all projects in this book. Once the location is set, you can skip this step in the future.

6. **Click the Next button.**

   The next screen (the final one) allows you to select a compiler and choose whether to create Debug or Release versions of your code, or both.

   The compiler selection is fine; the GNU GCC Compiler (or whatever is shown in the window) is the one you want.

7. **Remove the check mark by Create Debug Configuration.**

   You create this configuration only when you need to *debug,* or fix, a programming predicament that puzzles you. See Chapter 25.

8. **Click the Finish button.**

Code::Blocks builds a skeleton of your project, which is the same code shown in Listing 1-1. Skim to the later section "Building and running."

Create an empty file

A less complicated method for working with this book's exercises is to type the source code into an empty project file. Obey these steps in the Code::Blocks IDE:

1. **Choose File, New, Empty File.**
2. **Type the source code into the editor.**

   You can also copy-and-paste the source code from the companion website into the editor. Refer to the introduction for details on the companion website.


TIP

   If you'd prefer to see the editor color-code your text, save the file! Press Ctrl+S and choose the folder where you plan to keep all this book's programming projects. Name the file according to this book's convention: **ex** followed by a 2-digit chapter number and 2-digit project number. End the filename in **.c** ("dot C") to identify it as a C source code file.

3. **Save the source code file.**

   If the file is already saved, press Ctrl+S. Otherwise, follow the naming convention listed under Step 2.

After the empty file is created, you can build and run the program just as you would had you run through the bothersome technique of starting a Code::Blocks project.


TIP

You can also open the source code file directly in Code::Blocks, if you've obtained it from the companion website: Use the Open command to open the file. At this point, you can build and run, modify the code, or do whatever your heart pleases.

Building and running

In Code::Blocks, the process of compiling and linking C language source code is accomplished in one step called Build.

To build the project, click the Build button on the toolbar, as shown in the margin and illustrated in Figure 1-1. Any compiler warnings or errors appear on the Build Log tab at the bottom of the window. For Listing 1-1, no warnings or errors should be generated, providing the code is input exactly.

To test-run the program, click the Run button, shown in the margin and illustrated in Figure 1-1. Output appears in a terminal window, as shown in Figure 1-2.

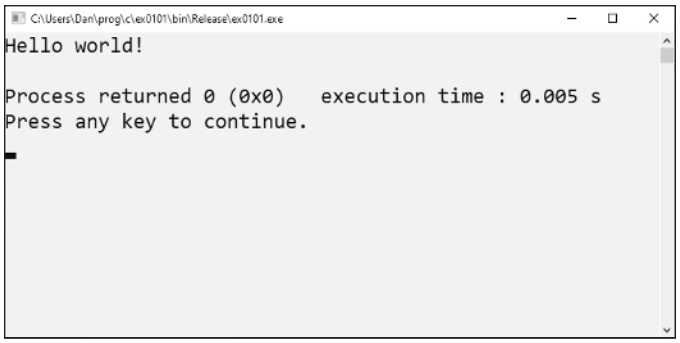

FIGURE 1-2: Program output.

Press the Enter key to close the output window. In Linux, you must type the **exit** command to close the window.

Both build and run steps are combined when you click the Build and Run button, shown in the margin.

When you're done with a project, or even after you've changed a minor thing, save it. Save the source code file, but also save the project if one was created: Choose File, Save Everything to save both the source code and project files.

- Commands for building and running the code are also found on the Build menu.
- The keyboard shortcuts for Build, Run, and Build and Run are Ctrl+F9, Ctrl+F10, and F9, respectively. No need to memorize those shortcuts; they're listed on the menu.

TIP

C LIBRARY DOCUMENTATION

You can view documentation for the various C library functions in two ways.

First, in a terminal window, use the *man* program to look up function definitions. For example, type *man printf* to read about the *printf()* function. This documentation is referred to as the "man pages," where "man" is short for *manual*.

The *man* program also documents shell commands. Occasionally, a shell command shares the name of a C language function, such as *stat*. To ensure that you're viewing the proper *man* page, use the *page* argument to specify either page 2 or page 3, where most of the C language functions are found. For example:

man stat — view the *man* page on the *stat* shell command.

man 2 stat — view the *man* page for the C language *stat()* function.

If you're not using a Unix-like shell to program, *man* page documentation is found on the Internet. I recommend these pages:

```
http://man7.org/linux/man-pages/dir_section_2.html

http://man7.org/linux/man-pages/dir_section_3.html
```

TECHNICAL STUFF

» The program output appears in the top part of the command prompt window (refer to Figure 1-2). The last two lines are generated by the IDE when the program is run. The text shows a value returned from the program to the operating system, a zero, and how long the program took to run (0.005 seconds).