Chap 1. Getting Started with C

1-1. A Brief History of the C Language

You might be wondering about the origin of the C language and where it got its name. C was created by Dennis Ritchie at the Bell Telephone Laboratories in 1972. The language wasn't created for the fun of it, but for a specific purpose: to design the UNIX operating system (which is used on many computers). From the beginning, C was intended to be useful--to allow busy programmers to get things done.

Because C is such a powerful and flexible language, its use quickly spread beyond Bell Labs. Programmers everywhere began using it to write all sorts of programs. Soon, however, different organizations began utilizing their own versions of C, and subtle differences between implementations started to cause programmers headaches. In response to this problem, the American National Standards Institute (ANSI) formed a committee in 1983 to establish a standard definition of C, which became known as ANSI Standard C. With few exceptions, every modern C compiler has the ability to adhere to this standard.

Now, what about the name? The C language is so named because its predecessor was called B. The B language was developed by Ken Thompson of Bell Labs. You should be able to guess why it was called B.

1-2. Why Use C?

In today's world of computer programming, there are many high-level languages to choose from, such as C, Pascal, BASIC, and Java. These are all excellent languages suited for most programming tasks. Even so, there are several reasons why many computer professionals feel that C is at the top of the list:

- C is a powerful and flexible language. What you can accomplish with C is limited only by your imagination. The language itself places no constraints on you. C is used for projects as diverse as operating systems, word processors, graphics, spreadsheets, and even compilers for other languages.
- C is a popular language preferred by professional programmers. As a result, a wide variety of C compilers and helpful accessories are available.
- C is a portable language. *Portable* means that a C program written for one computer system (an IBM PC, for example) can be compiled and run on another system (a DEC VAX system, perhaps) with little or no modification. Portability is enhanced by the ANSI standard for C, the set of rules for C compilers.
- C is a language of few words, containing only a handful of terms, called *keywords,* which serve as the base on which the language's functionality is built. You might think that a language with more keywords (sometimes called *reserved words*) would be more powerful. This isn't true. As you program with C, you will find that it can be programmed to do any task.
- C is modular. C code can (and should) be written in routines called *functions*. These functions can be reused in other applications or programs. By passing pieces of information to the functions, you can create useful, reusable code.

As these features show, C is an excellent choice for your first programming language. What about C++? You might have heard about C++ and the programming technique called *object-oriented programming*. Perhaps you're wondering what the differences are between C and C++ and whether you should be teaching yourself C++ instead of C.

Not to worry! C++ is a superset of C, which means that C++ contains everything C does, plus new additions for object-oriented programming. If you do go on to learn C++, almost everything you learn about C will still apply to the C++ superset. In learning C, you are not only learning one of today's most powerful and popular programming languages, but you are also preparing yourself for object-oriented programming.

Another language that has gotten lots of attention is Java. Java, like C++, is based on C. If later you decide to learn Java, you will find that almost everything you learned about C can be applied.

1-3. Preparing to Program

You should take certain steps when you're solving a problem. First, you must define the problem. If you don't know what the problem is, you can't find a solution! Once you know what the problem is, you can devise a plan to fix it. Once you have a plan, you can usually implement it. Once the plan is implemented, you must test the results to see whether the problem is solved. This same logic can be applied to many other areas, including programming.

When creating a program in C (or for that matter, a computer program in any language), you should follow a similar sequence of steps:

1. Determine the objective(s) of the program.
2. Determine the methods you want to use in writing the program.
3. Create the program to solve the problem.
4. Run the program to see the results.

An example of an objective (see step 1) might be to write a word processor or database program. A much simpler objective is to display your name on the screen. If you didn't have an objective, you wouldn't be writing a program, so you already have the first step done.

The second step is to determine the method you want to use to write the program. Do you need a computer program to solve the problem? What information needs to be tracked? What formulas will be used? During this step, you should try to determine what you need to know and in what order the solution should be implemented.

As an example, assume that someone asks you to write a program to determine the area inside a circle. Step 1 is complete, because you know your objective: determine the area inside a circle. Step 2 is to determine what you need to know to ascertain the area. In this example, assume that the user of the program will provide the radius of the circle. Knowing this, you can apply the formula pr^2 to obtain the answer. Now you have the pieces you need, so you can continue to steps 3 and 4, which are called the Program Development Cycle.

1-4. The Program Development Cycle

The Program Development Cycle has its own steps. In the first step, you use an editor to create a disk file containing your source code. In the second step, you compile the source code to create an object file. In the third step, you link the compiled code to create an executable file. The fourth step is to run the program to see whether it works as originally planned.

1-4-1. Creating the Source Code

Source code is a series of statements or commands that are used to instruct the computer to perform your desired tasks. As mentioned, the first step in the Program Development Cycle is to enter source code into an editor. For example, here is a line of C source code:

```
printf("Hello, Mom!");
```

This statement instructs the computer to display the message Hello, Mom! on-screen. (For now, don't worry about how this statement works.)

Using an Editor

Most compilers come with a built-in editor that can be used to enter source code; however, some don't. Consult your compiler manuals to see whether your compiler came with an editor. If it didn't, many alternative editors are available.

Most computer systems include a program that can be used as an editor. If you're using a UNIX system, you can use such editors as ed, ex, edit, emacs, or vi. If you're using Microsoft Windows, Notepad is available. If you're using MS/DOS 5.0 or later, you can use Edit. If you're using a version of DOS before 5.0, you can use Edlin. If you're using PC/DOS 6.0 or later, you can use E. If you're using OS/2, you can use the E and EPM editors.

Most word processors use special codes to format their documents. These codes can't be read correctly by other programs. The American Standard Code for Information Interchange (ASCII) has specified a standard text format that

nearly any program, including C, can use. Many word processors, such as WordPerfect, AmiPro, Word, WordPad, and WordStar, are capable of saving source files in ASCII form (as a text file rather than a document file). When you want to save a word processor's file as an ASCII file, select the ASCII or text option when saving.

If none of these editors is what you want to use, you can always buy a different editor. There are packages, both commercial and shareware, that have been designed specifically for entering source code.

NOTE: To find alternative editors, you can check your local computer store or computer mail-order catalogs. Another place to look is in the ads in computer programming magazines.

When you save a source file, you must give it a name. The name should describe what the program does. In addition, when you save C program source files, give the file a .C extension. Although you could give your source file any name and extension, .C is recognized as the appropriate extension to use.

1-4-2. Compiling the Source Code

Although you might be able to understand C source code (at least, after reading this book you will be able to), your computer can't. A computer requires digital, or *binary,* instructions in what is called *machine language*. Before your C program can run on a computer, it must be translated from source code to machine language. This translation, the second step in program development, is performed by a program called a *compiler*. The compiler takes your source code file as input and produces a disk file containing the machine language instructions that correspond to your source code statements. The machine language instructions created by the compiler are called *object code,* and the disk file containing them is called an *object file*.

NOTE: This book covers ANSI Standard C. This means that it doesn't matter which C compiler you use, as long as it follows the ANSI Standard.

Each compiler needs its own command to be used to create the object code. To compile, you typically use the command to run the compiler followed by the source filename. The following are examples of the commands issued to compile a source file called RADIUS.C using various DOS/Windows compilers:

Compiler	*Command*
Microsoft C	cl radius.c
Borland's Turbo C	tcc radius.c
Borland C	bcc radius.c
Zortec C	ztc radius.c

To compile RADIUS.C on a UNIX machine, use the following command:

```
cc radius.c
```

Consult the compiler manual to determine the exact command for your compiler.

If you're using a graphical development environment, compiling is even simpler. In most graphical environments, you can compile a program listing by selecting the compile icon or selecting something from a menu. Once the code is compiled, selecting the run icon or selecting something from a menu will execute the program. You should check your compiler's manuals for specifics on compiling and running a program.

After you compile, you have an object file. If you look at a list of the files in the directory or folder in which you compiled, you should find a file that has the same name as your source file, but with an .OBJ (rather than a .C) extension. The .OBJ extension is recognized as an object file and is used by the linker. On UNIX systems, the compiler creates object files with an extension of .O instead of .OBJ.

1-4-3. Linking to Create an Executable File

One more step is required before you can run your program. Part of the C language is a function library that contains *object code* (code that has already been compiled) for predefined functions. A *predefined function* contains C code that has already been written and is supplied in a ready-to-use form with your compiler package.

The printf() function used in the previous example is a *library function*. These library functions perform frequently needed tasks, such as displaying information on-screen and reading data from disk files. If your program uses any of these functions (and hardly a program exists that doesn't use at least one), the object file produced when your source code was compiled must be combined with object code from the function library to create the final executable program. (*Executable* means that the program can be run, or executed, on your computer.) This process is called *linking,* and it's performed by a program called (you guessed it) a *linker*.

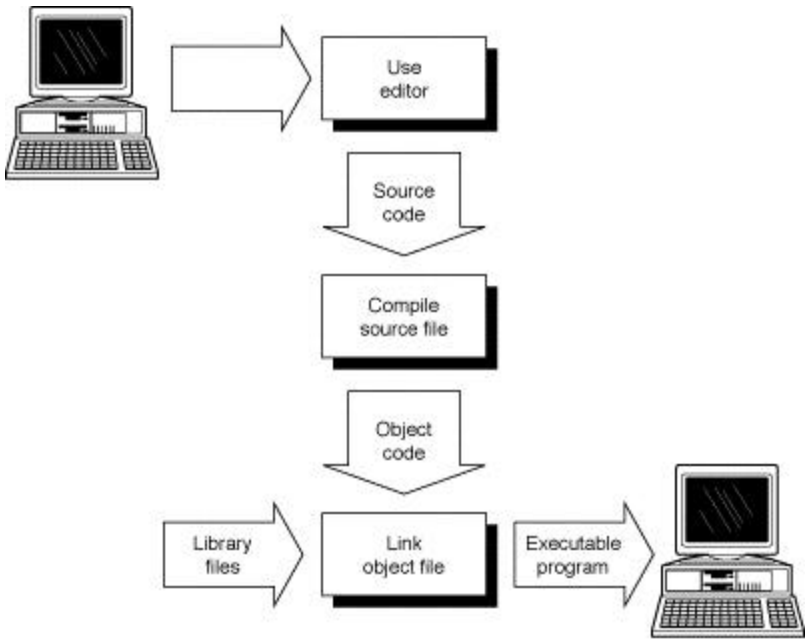

Figure 1. The C source code that you write is converted to object code by the compiler and then to an executable file by the linker.

1-4-4. Completing the Development Cycle

Once your program is compiled and linked to create an executable file, you can run it by entering its name at the system prompt or just like you would run any other program. If you run the program and receive results different from what you thought you would, you need to go back to the first step. You must identify what caused the problem and correct it in the source code. When you make a change to the source code, you need to recompile and relink the program to create a corrected version of the executable file. You keep following this cycle until you get the program to execute exactly as you intended.

One final note on compiling and linking: Although compiling and linking are mentioned as two separate steps, many compilers, such as the DOS compilers mentioned earlier, do both as one step. Regardless of the method by which compiling and linking are accomplished, understand that these two processes, even when done with one command, are two separate actions.

The C Development Cycle

Step 1	Use an editor to write your source code. By tradition, C source code files have the extension .C (for example, MYPROG.C, DATABASE.C, and so on).
Step	Compile the program using a compiler. If the compiler doesn't find any errors in the program,

2	it produces an object file. The compiler produces object files with an .OBJ extension and the same name as the source code file (for example, MYPROG.C compiles to MYPROG.OBJ). If the compiler finds errors, it reports them. You must return to step 1 to make corrections in your source code.
Step 3	Link the program using a linker. If no errors occur, the linker produces an executable program located in a disk file with an .EXE extension and the same name as the object file (for example, MYPROG.OBJ is linked to create MYPROG.EXE).
Step 4	Execute the program. You should test to determine whether it functions properly. If not, start again with step 1 and make modifications and additions to your source code.

For all but the simplest programs, you might go through this sequence many times before finishing your program. Even the most experienced programmers can't sit down and write a complete, error-free program in just one step! Because you'll be running through the edit-compile-link-test cycle many times, it's important to become familiar with your tools: the editor, compiler, and linker.

1-5. Your First C Program

You're probably eager to try your first program in C. To help you become familiar with your compiler, here's a quick program for you to work through. You might not understand everything at this point, but you should get a feel for the process of writing, compiling, and running a real C program.

This demonstration uses a program named HELLO.C, which does nothing more than display the words Hello, World! on-screen. This program, a traditional introduction to C programming, is a good one for you to learn. The source code for HELLO.C is in Listing 1.1. When you type in this listing, you won't include the line numbers or colons.

Listing 1.1. HELLO.C.

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("Hello, World!\n");
6:     return 0;
7: }
```

Be sure that you have installed your compiler as specified in the installation instructions provided with the software. Whether you are working with UNIX, DOS, or any other operating system, make sure you understand how to use the compiler and editor of your choice. Once your compiler and editor are ready, follow these steps to enter, compile, and execute HELLO.C.

1-5-1. Entering and Compiling HELLO.C

To enter and compile the HELLO.C program, follow these steps:

1. Make active the directory your C programs are in and start your editor. As mentioned previously, any text editor can be used, but most C compilers (such as Borland's Turbo C++ and Microsoft's Visual C/C++) come with an integrated development environment (IDE) that lets you enter, compile, and link your programs in one convenient setting. Check the manuals to see whether your compiler has an IDE available.
2. Use the keyboard to type the HELLO.C source code exactly as shown in Listing 1.1. Press Enter at the end of each line.

NOTE: Don't enter the line numbers or colons. These are for reference only.

---

3. Save the source code. You should name the file HELLO.C.
4. Verify that HELLO.C is on disk by listing the files in the directory or folder. You should see HELLO.C within this listing.
5. Compile and link HELLO.C. Execute the appropriate command specified by your compiler's manuals. You should get a message stating that there were no errors or warnings.
6. Check the compiler messages. If you receive no errors or warnings, everything should be okay.
If you made an error typing the program, the compiler will catch it and display an error message. For example, if you misspelled the word printf as prntf, you would see a message similar to the following:

```
Error: undefined symbols:_prntf in hello.c (hello.OBJ)
```

7. Go back to step 2 if this or any other error message is displayed. Open the HELLO.C file in your editor. Compare your file's contents carefully with Listing 1.1, make any necessary corrections, and continue with step 3.
8. Your first C program should now be compiled and ready to run. If you display a directory listing of all files named HELLO (with any extension), you should see the following:
HELLO.C, the source code file you created with your editor
HELLO.OBJ or HELLO.O, which contains the object code for HELLO.C
HELLO.EXE, the executable program created when you compiled and linked HELLO.C
9. To *execute,* or run, HELLO.EXE, simply enter hello. The message Hello, World! is displayed on-screen.

Congratulations! You have just entered, compiled, and run your first C program. Admittedly, HELLO.C is a simple program that doesn't do anything useful, but it's a start. In fact, most of today's expert C programmers started learning C in this same way--by compiling HELLO.C--so you're in good company.

Compilation Errors

A compilation error occurs when the compiler finds something in the source code that it can't compile. A misspelling, typographical error, or any of a dozen other things can cause the compiler to choke. Fortunately, modern compilers don't just choke; they tell you what they're choking on and where it is! This makes it easier to find and correct errors in your source code.

This point can be illustrated by introducing a deliberate error into HELLO.C. If you worked through that example (and you should have), you now have a copy of HELLO.C on your disk. Using your editor, move the cursor to the end of the line containing the call to printf(), and erase the terminating semicolon. HELLO.C should now look like Listing 1.2.

Listing 1.2. HELLO.C with an error.

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf("Hello, World!")
6:     return 0;
7: }
```

Next, save the file. You're now ready to compile it. Do so by entering the command for your compiler. Because of the error you introduced, the compilation is not completed. Rather, the compiler displays a message similar to the following:

```
hello.c(6) : Error: `;' expected
```

Looking at this line, you can see that it has three parts:

hello.c The name of the file where the error was found

(6) : The line number where the error was found

Error: `;' expected A description of the error

This message is quite informative, telling you that in line 6 of HELLO.C the compiler expected to find a semicolon but didn't. However, you know that the semicolon was actually omitted from line 5, so there is a discrepancy. You're faced with the puzzle of why the compiler reports an error in line 6 when, in fact, a semicolon was omitted from line 5. The answer lies in the fact that C doesn't care about things like breaks between lines. The semicolon that belongs after the printf() statement could have been placed on the next line (although doing so would be bad programming practice). Only after encountering the next command (return) in line 6 is the compiler sure that the semicolon is missing. Therefore, the compiler reports that the error is in line 6.

This points out an undeniable fact about C compilers and error messages. Although the compiler is very clever about detecting and localizing errors, it's no Einstein. Using your knowledge of the C language, you must interpret the compiler's messages and determine the actual location of any errors that are reported. They are often found on the line reported by the compiler, but if not, they are almost always on the preceding line. You might have a bit of trouble finding errors at first, but you should soon get better at it.

NOTE: The errors reported might differ depending on the compiler. In most cases, the error message should give you an idea of what or where the problem is.

Before leaving this topic, let's look at another example of a compilation error. Load HELLO.C into your editor again and make the following changes:

1. Replace the semicolon at the end of line 5.
2. Delete the double quotation mark just before the word Hello.

Save the file to disk and compile the program again. This time, the compiler should display error messages similar to the following:

```
hello.c(5) : Error: undefined identifier `Hello'
hello.c(7) : Lexical error: unterminated string
Lexical error: unterminated string
Lexical error: unterminated string
Fatal error: premature end of source file
```

The first error message finds the error correctly, locating it in line 5 at the word Hello. The error message undefined identifier means that the compiler doesn't know what to make of the word Hello, because it is no longer enclosed in quotes. However, what about the other four errors that are reported? These errors, the meaning of which you don't need to worry about now, illustrate the fact that a single error in a C program can sometimes cause multiple error messages.

The lesson to learn from all this is as follows: If the compiler reports multiple errors, and you can find only one, go ahead and fix that error and recompile. You might find that your single correction is all that's needed, and the program will compile without errors.

Linker Error Messages

Linker errors are relatively rare and usually result from misspelling the name of a C library function. In this case, you get an Error: undefined symbols: error message, followed by the misspelled name (preceded by an underscore). Once you correct the spelling, the problem should go away.

1-6. Summary

After reading this chapter, you should feel confident that selecting C as your programming language is a wise choice. C offers an unparalleled combination of power, popularity, and portability. These factors, together with C's close relationship to the C++ object-oriented language as well as Java, make C unbeatable.

This chapter explained the various steps involved in writing a C program--the process known as program development. You should have a clear grasp of the edit-compile-link-test cycle, as well as the tools to use for each step.

Errors are an unavoidable part of program development. Your C compiler detects errors in your source code and displays an error message, giving both the nature and the location of the error. Using this information, you can edit your source code to correct the error. Remember, however, that the compiler can't always accurately report the nature and location of an error. Sometimes you need to use your knowledge of C to track down exactly what is causing a given error message.

Q&A

Q If I want to give someone a program I wrote, which files do I need to give him?
A One of the nice things about C is that it is a compiled language. This means that after the source code is compiled, you have an executable program. This executable program is a stand-alone program. If you wanted to give HELLO to all your friends with computers, you could. All you need to give them is the executable program, HELLO.EXE. They don't need the source file, HELLO.C, or the object file, HELLO.OBJ. They don't need to own a C compiler, either.
Q After I create an executable file, do I need to keep the source file (.C) or object file (.OBJ)?
A If you get rid of the source file, you have no way to make changes to the program in the future, so you should keep this file. The object files are a different matter. There are reasons to keep object files, but they are beyond the scope of what you're doing now. For now, you can get rid of your object files once you have your executable file. If you need the object file, you can recompile the source file.
Most integrated development environments create files in addition to the source file (.C), the object file (.OBJ or .O), and the executable file. As long as you keep the source file (.C), you can always recreate the other files.
Q If my compiler came with an editor, do I have to use it?
A Definitely not. You can use any editor, as long as it saves the source code in text format. If the compiler came with an editor, you should try to use it. If you like a different editor better, use it. I use an editor that I purchased separately, even though all my compilers have their own editors. The editors that come with compilers are getting better. Some of them automatically format your C code. Others color-code different parts of your source file to make it easier to find errors.
Q Can I ignore warning messages?
A Some warning messages don't affect how the program runs, and some do. If the compiler gives you a warning message, it's a signal that something isn't right. Most compilers let you set the warning level. By setting the warning level, you can get only the most serious warnings, or you can get all the warnings, including the most minute. Some compilers even offer various levels in-between. In your programs, you should look at each warning and make a determination. It's always best to try to write all your programs with absolutely no warnings or errors. (With an error, your compiler won't create the executable file.)

Workshop

The Workshop provides quiz questions to help you solidify your understanding of the material covered and exercises to provide you with experience in using what you've learned. Try to understand the quiz and exercise answers before continuing to the next chapter. Answers are provided in Appendix G, "Answers."

Quiz

1. Give three reasons why C is the best choice of programming language.
2. What does the compiler do?
3. What are the steps in the program development cycle?
4. What command do you need to enter in order to compile a program called PROGRAM1.C with your compiler?
5. Does your compiler do both the linking and compiling with just one command, or do you have to enter separate commands?
6. What extension should you use for your C source files?
7. Is FILENAME.TXT a valid name for a C source file?
8. If you execute a program that you have compiled and it doesn't work as you expected, what should you do?
9. What is machine language?
10. What does the linker do?

Exercises

1. Use your text editor to look at the object file created by Listing 1.1. Does the object file look like the source file? (Don't save this file when you exit the editor.)
2. Enter the following program and compile it. What does this program do? (Don't include the line numbers or colons.)

```
1: #include <stdio.h>
2:
3: int radius, area;
4:
5: main()
6: {
7:     printf( "Enter radius (i.e. 10): " );
8:     scanf( "%d", &radius );
9:     area = (int) (3.14159 * radius * radius);
10:    printf( "\n\nArea = %d\n", area );
11:    return 0;
12: }
```

3. Enter and compile the following program. What does this program do?

```
1: #include <stdio.h>
2:
3: int x,y;
4:
5: main()
6: {
7:     for ( x = 0; x < 10; x++, printf( "\n" ) )
8:         for ( y = 0; y < 10; y++ )
9:             printf( "X" );
10:
11:    return 0;
12: }
```

4. BUG BUSTER: The following program has a problem. Enter it in your editor and compile it. Which lines generate error messages?

```
1: #include <stdio.h>
2:
3: main();
4: {
5:     printf( "Keep looking!" );
6:     printf( "You\'ll find it!\n" );
```

```
7:     return 0;
8: }
```

5. BUG BUSTER: The following program has a problem. Enter it in your editor and compile it. Which lines generate problems?

```
1: #include <stdio.h>
2:
3: main()
4: {
5:     printf( "This is a program with a " );
6:     do_it( "problem!");
7:     return 0;
8: }
```

6. Make the following change to the program in exercise 3. Recompile and rerun this program. What does the program do now?

```
9: printf( "%c", 1 );
```