Sams Teach Yourself C in 24 Hours

Hour 2 - Writing Your First C Program

Cut your own wood and it will warm you twice.

—Chinese proverb

In Hour 1, "Getting Started," you learned that C is a high-level programming language and that you need a C compiler to translate your C programs into binary code that your computer can understand and execute. In this lesson you'll learn to write your first C program and the basics of a C program, such as

- The #include directive
- Header files
- Comments
- The main() function
- The return statement
- The exit() function
- The newline character (\n)
- The void data type
- Translating a C program into an executable file
- Debugging

A Simple C Program

Let's have a look at our first C program, demonstrated in Listing 2.1. Later in this lesson you're going to write your own C program for the first time.

TYPE
Listing 2.1. A simple C program.

```
1:  /* 02L01.c: This is my first C program */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     printf ("Howdy, neighbor! This is my first C program.\n");
7:     return 0;
8:  }
```

This is a very simple C program, which is saved in a file called 02L01.c. Note that the name of a C program file must have an extension of .c. If you've installed a C compiler and set up the proper development environment, you should be able to compile this C program and make it into an executable file.

I set up my development environment in such a way that all C programs in this book can be compiled and made into DOS-based applications. For instance, 02L01.exe is the name of the DOS application made from 02L01.c. Note that .exe is included as the extension to the name of a DOS application program (that is, an executable file).

Also, on my machine, I save all the executable files made from the C programs in this book into a dedicated directory called C:\app. Therefore, if I type in 02L01 from a DOS prompt and press the Enter key, I can run the 02L01.exe executable file and display the message Howdy, neighbor! This is my first C program. on the screen. The following output is a copy from the screen:

```
C:\app> 02L01
Howdy, neighbor! This is my first C program.
C:\app>
```

Comments

Now let's take a close look at the C program in Listing 2.1.

The first line contains a comment:

```
/* 02L01.C: This is my first C program */
```

You notice that this line starts with a combination of a slash and an asterisk, /*, and ends with */. In C, /* is called the opening comment mark, and */ is the closing comment mark. The C compiler ignores everything between the opening comment mark and closing comment mark. That means the comment in the first line of Listing 2.1, 02L01.C: This is my first C program, is ignored by the compiler.

The only purpose of including comments in your C program is to help you document what the program or some specific sections in the programs do. Remember, comments are written for programmers like you. For example, when you read someone's code, the comments in the code help you to understand what the code does, or at least what the code intends to do.

You don't need to worry about the size or performance speed of your C program if you add many comments into it. Adding comments into a C program does not increase the size of the binary code of the program (that is, the executable file), although the size of the program itself (that is, the source code) may become larger. Also, the performance speed of the executable file made from your C program is not affected by the comments inside your C program.

Most C compilers allow you to write a comment that crosses more than one line. For instance, you can write a comment in C like this:

```
/*
   This comment does not increase the size of
   the executable file (binary code), nor does
   it affect the performance speed.
*/
```

which is equivalent to this:

```
/* This comment does not increase the size of */
/* the executable file (binary code), nor does */
/* it affect the performance speed. */
```

NOTE

These days, there is another way to put comments into a C program. C++ started using two slashes (//) to mark the beginning of a comment line; many C compilers now use this convention as well. The comment ends at the end of the line. For instance, if I write a C program in Borland C++ or Visual C++, the following two comments are identical:

```
/*
   This comment does not increase the size of
   the executable file (binary code), nor does
   it affect the performance speed.
*/
// This comment does not increase the size of
// the executable file (binary code), nor does
// it affect the performance speed.
```

Note that this new style of using // as the beginning mark of a comment has not been approved by ANSI. Make sure your C compiler supports // before you use it.

One thing that needs to be pointed out is that the ANSI standard does not support nested comments, that is, comments within comments. For instance, the following is not allowed by the ANSI standard:

```
/* This is the first part of the first comment
   /* This is the second comment */
   This is the second part of the first comment */
```

TIP

You can use the opening comment mark, /*, and closing comment mark, */, to help you test and fix any errors found in your C program. You can comment out one or more C statements in your C program with /* and */ when you need to focus on other statements and watch their behaviors closely. The C compiler will ignore the statements you comment out.

Later, you can always restore the previously commented-out statements simply by removing the opening comment and closing comment marks. In this way, you don't need to erase or rewrite any statements during the testing and debugging.

The #include Directive

Let's now move to line 2 in the C program of Listing 2.1:

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

You see that this line starts with a pound sign, #, which is followed by include. In C, #include forms a preprocessor directive that tells the C preprocessor to look for a file and place the contents of the file in the location where the #include directive indicates.

The preprocessor is a program that does some preparations for the C compiler before your code is compiled. More details about the C preprocessor are discussed in Hour 23, "The C Preprocessor."

Also in this line, you see that <stdio.h> follows #include. You may guess that the file the #include directive asks for is something called stdio.h. You are exactly right! Here, the #include directive does ask the C preprocessor to look for and place stdio.h where the directive is in the C program.

The name of the stdio.h file stands for standard input-output header file. The stdio.h file contains numerous prototypes and macros to perform input or output (I/O) for C programs. You'll see more program I/O in Hour 5, "Reading from and Writing to Standard I/O."

NOTE

The C programming language distinguishes between lowercase and uppercase characters. In other words, C is a case-sensitive language. For instance, stdio.h and STDIO.H are different filenames in C. Likewise, main() and Main() are two different function names.

Header Files

The files that are required by the #include directive, like stdio.h, are called header files because the #include directives are almost always placed at the head of C programs. Actually, the extension name of .h does mean "header."

Besides stdio.h, there are more header files, such as stdlib.h, string.h, math.h, and so on. Appendix A, "ANSI Standard Header Files," gives a list of all the ANSI standard header files.

Angle Brackets (< >) and Double Quotes (" ")

In the second line of Listing 2.1, there are two angle brackets, < and >, that are used to surround stdio.h. You may be wondering what the angle brackets do. In C, the angle brackets ask the C preprocessor to look for a header file in a directory other than the current one.

For instance, the current directory containing the 02L01.C file is called C:\code on my computer. Therefore, the angle brackets around <stdio.h> tell the C preprocessor to look for stdio.h in a directory other than C:\code.

If you want to let the C preprocessor look into the current directory first for a header file before it starts to look elsewhere, you can use double quotes to surround the name of the header file. For instance, when the C preprocessor sees "stdio.h", it looks in the current directory, which is C:\code on my machine, first before it goes elsewhere for the stdio.h header file.

Normally, the header files are saved in a subdirectory called include. For instance, I might install a Microsoft C compiler in the directory MSVC on my hard drive, which is labeled as the C drive. Then the path to access the header files becomes C:\MSVC\include.

The main() Function

In line 4 of Listing 2.1, you see this function:

```
main ()
```

This is a very special function in C. Every C program must have a main() function, and every C program can only have one main() function. More generic discussions about functions are given in Hour 3, "The Essentials of C Programs."

You can put the main() function wherever you want in your C program. However, the execution of your program always starts with the main() function.

In Listing 2.1, the main() function body starts in line 4 and ends in line 8. Because this is a very simple program, the main() function is the only function defined in the program. Within the main() function body, a C library function, printf(), is called in order to print out a greeting message. (See line 6.) More details about printf() are covered in Hour 5.

One more important thing about main() is that the execution of every C program ends with main(). A program ends when all the statements within the main() function have been executed.

The Newline Character (\n)

In the printf() function, one thing worth mentioning at this moment is the newline character, \n. Usually suffixed at the end of a message, the newline character tells the computer to generate a carriage-return and line-feed sequence so that anything printed out after the message will start on the next new line on the screen.

Exercise 3 in this lesson gives you a chance to use the newline character to break a one-line message into two lines.

The return Statement

All functions in C can return values. For instance, when you make a function to add two numbers, you can make such a function that returns to you the value of the addition.

The main() function itself returns a value. By default, main() returns an integer. In C, integers are decimal numbers without fraction portions.

Therefore, in line 7 of Listing 2.1, there is a statement, return 0;, that indicates that 0 is returned from the main() function and the program is terminated normally.

A nonzero value returned by the return statement tells the operating system that an error has occurred. The bigger the return value, the more severe the error.

The exit() Function

There is also a C library function, exit(), that can be used to cause a program to end. Because the exit() function is defined in a header file, stdlib.h, you have to include the header file at the beginning of your program.

Unlike main(), the exit() function itself does not return any values, but the argument to exit() indicates whether the program is terminated normally. A nonzero argument to the exit() function tells the operating system that the program has terminated abnormally.

Actually, you can replace return 0; in line 7 of Listing 2.1 with exit(0); and get a similar result after running the modified program.

Note that return and exit() can also be used in other functions. You'll see more examples in the rest of the book.

Listing 2.2 contains the program that uses exit() instead of return.

TYPE
Listing 2.2. A C program with exit().

```
1:  /* 02L02.c */
2:  #include <stdlib.h>
3:  #include <stdio.h>
4:
5:  void main()
6:  {
7:     printf ("Howdy, neighbor! This is my first C program.\n");
8:     exit(0);
9:  }
```

After compiling the program in Listing 2.2, you should be able to run the program and get the same message, Howdy, neighbor! This is my first C program., printed out on the screen.

The void Data Type

You may notice that the void word has been added into the C program in Listing 2.2. void is a keyword for a data type in C. When a void is placed prior to a function name, it indicates that the function does not return a value.

As you have learned, the exit() function does not return any values, but, by default, the main() function does. Therefore, as shown in line 5 of Listing 2.2, void is used to modify the returning data type of main() and to make the main() function not return any value. (You'll learn more about data types in C in Hours 4, "Data Types and Names in C," and 18, "More Data Types and Functions.")

Compiling and Linking

You may already be anxious to know how an executable file is made. Let's have a look at how a C program is compiled and translated into an executable file (binary code). As shown in Figure 2.1, there are at least three steps needed to create an executable file.

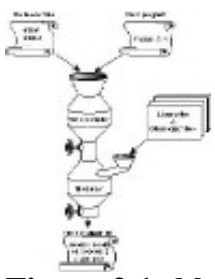

Figure 2.1. *Make an executable file by the compiler and linker.*

First, a program written in C, called source code, is made. Then the source code is compiled by a C compiler, which creates a new file. The new file is an object file. In the UNIX operating system, the name of an object file ends with the extension .o; in the DOS operating system, the extension is .obj.

You cannot execute the object file because there is some function code missing. You have to finish the next step: linking. Linking is done by invoking a special program called a linker, which normally comes with the compiler package.

A linker is used to link together the object file, the ANSI standard C library, and other user-generated libraries to produce an executable file —the binary code. In this stage, the binary code of the library functions that are called in the source code is combined with the object file; the result is saved into a new file—an executable file. As you learned in the first hour of this book, the name of an executable file usually ends with the extension .exe in DOS.

(.com is another extension used for a DOS executable filename.) In UNIX, it's not necessary to include such an extension to an executable filename.

Later, you'll learn that in many cases, there may be several object files that have to be linked together in order to make an executable program.

Note that the object file and executable file are both machine-dependent. You cannot simply move an executable file, without recompiling the source code, from the current computer platform to another one that is operated by a different operating system even though the source code of the executable file, presumably written in ANSI C, is machine independent (that is, portable).

What's Wrong with My Program?

When you finish writing a C program and start to compile it, you might get some error or warning messages. Don't panic when you see error messages. We're human beings. Everybody makes mistakes. Actually, you should appreciate that your compiler catches some errors for you before you go any further.

Usually, your compiler can help you check the grammar of your C program and make sure you've followed the C programming rules properly. For instance, if you forget to put the ending brace on the main() function in line 8 of Listing 2.1, you'll get an error message something like this: syntax error : end of file found.

Also, the linker will issue an error message if it cannot find the missing code for a needed function in the libraries. For instance, if you misspell printf() as pprintf() in the program of Listing 2.1, you'll see an error message: `_pprintf': unresolved external (or something similar).

All errors found by the compiler and linker must be fixed before an executable file (binary code) can be made.

Debugging Your Program

In the computer world, program errors are also called bugs. In many cases, your C compiler and linker do not find any errors in your program, but the result generated by running the executable file of the program is not what you expect. In order to find those "hidden" errors in your program, you may need to use a debugger.

Normally, your C compiler vendor already includes a debugger software program in the C compiler package. The debugger can execute your program one line at a time so that you can watch closely what's going on with the code in each line, or so that you can ask the debugger to stop running your program on any line. For more details about your debugger, refer to the manuals made by your C compiler vendor.

Later in this book, you'll learn that debugging is a very necessary and important step in writing software programs. (This topic is covered in Hour 24, "What You Can Do Now.")

Summary

In this lesson you've learned the following:

- Some header files should be included at the beginning of your C program.
- Header files, such as stdio.h and stdlib.h, contain the declarations for functions used in your C program; for example, the printf() and exit() functions.
- Comments in your C programs are needed to help you document your programs. You can put comments anywhere you like in your programs.
- In ANSI C, a comment starts with the opening comment mark, /*, and ends with the closing comment mark, */.

- Every C program should have one but only one main() function. The program execution starts and ends with the main() function.
- The sequence of a carriage return and a line feed is carried out when the computer sees the newline character, \n.
- The return statement can be used to return a value to indicate to the operating system whether an error has occurred. The exit() function terminates a program; the argument to the function indicates the error status, too.
- The void data type can be used to modify the type of a return value for a function. Applying void to main() tells the operating system that the main() function does not return any value after termination.
- Compiling and linking are consecutive steps that have to be finished before an executable file is produced.
- Everybody, including yourself, makes mistakes in programming. Debugging is a very important step in your program design and coding.

In the next lesson you'll learn more about the essentials of C programs.

Q&A

Q Why do you need to put comments into your programs?

A Comments help us document what a program, or a special portion of a program, does. Especially when a program becomes very complex, we need to write comments to mark different parts in the program.

Q Why is the main() function needed in your program?

A The execution of a C program starts and ends with the main() function. Without the main() function, the computer does not know where to start to run a program.

Q What does the #include directive do?

A The #include directive is used to include header files that contain the declarations to the functions used in your C program. In other words, the #include directive tells the C preprocessor to look into directories and find the specified header file.

Q Why do you need a linker?

A After compiling, some function code may still be missing in the object file of a program. A linker must then be used to link the object file to the C standard library or other user-generated libraries and include the missing function code so that an executable file can be created.

Workshop

To help solidify your understanding of this hour's lesson, you are encouraged to answer the quiz questions and finish the exercises provided in the Workshop before you move to the next lesson. The answers and hints to the questions and exercises are given in Appendix E, "Answers to Quiz Questions and Exercises."

Quiz

1. Can a C compiler see the comments within your C program?
2. What kinds of files does a C compiler actually produce?
3. Does the exit() function return a value? How about the return statement?
4. What is a header file?

Exercises

1. Is #include <stdio.h> the same as #include "stdio.h"?
2. It's time for you to write your own first program. Referring to the program in Listing 2.1, write a C program that can print out a message: It's fun to write my own program in C.
3. Update the program in Listing 2.1 by adding one more newline character into the message printed out by the printf() function. You should see two lines of the message on the screen after running the updated executable file:

```
Howdy, neighbor!
This is my first C program.
```

4. What warning or error messages will you get when you're trying to compile the following program?

```
#include <stdlib.h>
#include <stdio.h>
main()
{
   printf ("Howdy, neighbor! This is my first C program.\n");
   exit(0);
}
```

5. What error messages will you get when you're trying to compile the following program?

```
void main()
{
   printf ("Howdy, neighbor! This is my first C program.\n");
   return 0;
}
```