CSC 111: Introduction to programming

Introduction

The programming language C was originally developed by Dennis Ritchie of Bell Laboratories and was designed to run with a UNIX operating system.

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.

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.

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.

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. 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.

Compiling the Source Code

Although you might be able to understand C source code, 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.***

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 **a linker.**

Your First 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;
}
```

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.

```
#include <stdio.h>

main()
{
    printf("Hello, World!")
    return 0;
}
```

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.

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.

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.

PROGRAMMING EXERCISES

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: }
```