Programming is fun. And programming in C is even more fun!

The best way to learn the C Programming language is by EXAMPLE!

You will notice lots of examples throughout this course. In addition to ME providing YOU examples, YOU should attempt to edit, compile, link and run the sample code. This way you can get lots of practice with coding and debugging. This is what we call "hands-on" learning. It is the only way you will truly learn how to program.

PROGRAM 2.1
(almost)

Let's take a look at (dissect) this program in the textbook. Although it looks simple enough, it illustrates a lot of very important programming concepts of the C language.

The program looks as follows. I have included line numbers which are NOT part of the code. I have included them as to make it easier to explain each line of code in the program.

```
1.  #include <stdio.h>
2.  void main(void)
3.  {
4.      printf("Programming is fun.\n" );
5.      getchar( );
6.  }
```

The results (output) of this program, after you edit it (type it in), compile it, link it, and then run it, would be:

```
Programming is fun.
```

Note: With most compiler environments your output will be sent to some sort of 'output screen' (window), and you may need to figure out how to get back to that output screen to view your results. Your compiler should have information for you as to how to view the output of your programs (like pressing F5).

With C-Free you will always see the output screen, and you will also get: ***Press any key to continue . . .*** which is not typical of other compilers.

Now, let's take a closer look at the program above.

Line 1: **#include <stdio.h>**

You should include this statement at the beginning of every C program. It tells the compiler that we plan to use one or more of the general functions in the **Standard C Library**. We will discuss the Standard C Library in more detail in a later topic within this lesson. Also, we discuss functions in great detail during Week 9.

In this case, the predefined function that we are using in the above program is the **printf** function, which is the **output** function in C.

Actually, what happens is the compiler automatically initiates a separate program called the **preprocessor**. The preprocessor looks for a # character in the first column of each line. If found, the preprocessor executes the statement. Because of this, the # character **must** be in the **first column** in the text file. You can have many of these preprocessor commands, but each must be on its own line in your file, and must begin with a # in column 1. We do not discuss the preprocessor per-se in this course, however, if you would like more information regarding the preprocessor, please refer to Chapter 14 in the textbook.

So, as far as you are concerned, at this point, all you have to know is that the above statement (line 1 in the code) must be included in every one of your C programs.

Line 2: **void main(void)**

This statement **must** be included in every C program. It (**main**) tells the CPU (processor) where program execution begins. The first '**void**' could be '**int**' instead, as in the text book (more on this later...).

The parentheses following the word **main ( )** indicates that *main* is the start of a function. Regarding functions: at this point, you should just realize that **every C program contains one or more functions, one of which must be main**. This is a very important concept in C programming.

All of the code written in C must reside within one or more functions. Again, we discuss functions in more detail in week 9.

Also, the words: **void** are very much related to functions, and will be discussed in greater detail in week 9. Some compilers complain about the first word void, and this may have to be omitted. Again, more on that later.....

At this point you're probably saying to yourself: *This stuff is easy!.* As well you should be. Because, guess what??? It is easy!!!!

Lines 3 and 6: **{** and **}**

These are the **begin** and **end** characters. They indicate that the segment of code between them constitute a **block of code**. In this case, they define a **function block**. Every function which is defined (in this case, function **main**), must have **begin** and **end** indicators. This simple requirement helps to keep your code structured. (Note, in the Pascal programming language, another strongly structured language, you actually type in the words BEGIN and END instead of using the braces.) We use the **begin** and end characters for a variety of reasons in C. At this point, all you have to know is that when you define a function, you must surround the function's code within these lovely braces! :-)

Also, how can I forget, please remember to **indent** the code within a block (braces) by 2 to 10 spaces. This greatly improves the readability of your program. Once you pick a number of spaces to indent, please use that number throughout your entire program, for consistency.

Line 4: **printf(" Programming in C is fun.\n");**

Well, as you can imagine, this line is the heart of the program. Let's take this one step at a time. Although it appears somewhat complex, you will soon realize that this line of code, like all others so far, is quite logical. (At least I hope so.)

First of all the **printf** statement: **printf** is a function (a segment of code) that is provided to us when we purchase a C compiler. It is a part of the Standard C Library group of functions. It is an output function whose purpose is to send text to the output device (monitor/screen).

In this statement, function **main()** is actually making a **function call** to function **printf().** When we make a function call, the function name is used (**printf**). Following the function name, you need to enclose function information (a.k.a. function arguments or function parameters) between parenthesis. In this case, function **printf** expects to receive a string of text (a group of characters) as function information. Function **printf** will then send the string of text to the output device (monitor). For this reason, we enclose the information (string of text) within the parenthesis. Also, notice that string of text is enclosed within quotation marks. This also is a requirement of the **printf** function.

So, if you accidentally typed in the above line of code as:

```
printf (Programming is fun.\n);
```

Then you would get a ***compiler*** or ***syntax*** error during compilation because you forgot to surround the text in quotation marks! This is tedious, yes, but remember the compiler expects things a certain way, and you must follow the rules. You will quickly learn that all of these very tedious rules are actually a good thing, as it ensures that you as a programmer is coding exactly as you intend to code. (I try telling my kids that all of our rules at home are actually a good thing, but they don't buy it either! :-))

So, when the printf function is called with the above line of code (Line 4), it will output each and every character it finds between the quotation marks, EXACTLY as you type them in. It does this for all characters in the string, but will stop when it reaches the **** (backslash) character.

The **backslash** character is called the **escape character** in C. It is an indicator to the printf function to temporarily halt the outputting of standard characters and to output something else. Depending on what character follows the backslash, is what else will be output. In our example, the character **n** follows the backslash. The entire sequence **\n** is considered an **escape sequence.** In C, the **\n** escape sequence tells the printf statement to output a carriage return and line feed (a.k.a. the ***new-line*** character). So, when the printf statement reaches that particular portion of the string, it will simply send a new line character to the output device. This is why when you look at the output above, you see that the prompt is displayed on the line *following* the text. It is the **\n** that caused the new line to be output causing the prompt to be moved to the next line.

Look at Table 9.2 on page 215 in your textbook.

This table contains a list of valid escape characters in C. Any of these can be included in your printf statements to output a variety of non-standard characters. For example, if you want an audible alarm (bell) to be output with your text, you simply include a \a in your printf statement. For example:

```
printf (" \a Invalid data entered.\n Please try again.\n" );
```

First, you would hear a ‘bell’, and at the same time you would see the following on your display:

```
 Invalid data entered.
```

Please try again.

Notice that you can embed escape sequences within the text in your printf function call. They do not have to be just at the beginning, or at the end of the text string. (You may not really hear a 'bell' ring – that was for the old CRT terminals.)

Finally, notice that the call to function printf ends with a semicolon (**;**). All C statements, within a function block, must be terminated by a semicolon. If you leave the semicolon out, you will get a compiler error.

Line 5: **getchar();**

Some compilers will 'flash' the output screen so fast that the user cannot see it. If you suspect this is happening with your program, insert a **getchar();** statement into the 'end' of your program.

We have already covered some of the basic fundamentals of a C program. You are well on your way! If you did not understand any of the concepts discussed so far, please post any/all questions to the discussion board.

We will now make a slight modification to the above program to illustrate some additional concepts of C.

PROGRAM 2.2

This program is similar to the prior one, but it includes an additional line of code in function main(). The code looks as follows:

```
#include <stdio.h>
int main(void)
{
   printf(" Programming is fun.\n" );
   printf("And programming in C is even more fun.\n" );
   getchar();
   return 0;
 }
```

The statement of code: **return 0;** is needed when the main program is type 'int' as above. The ***return*** statement will be explained in more detail when we discuss functions in Chapter 9.

So, as you can imagine, the output of the above program would look like:

```
Programming is fun.
And programming in C is even more fun.
```

I want you to notice 2 things here:

1. Each statement in the function block of function main was indented, and each was terminated with a semicolon.

2. If you left out the '\n' at the end of the first statement, BOTH of the above lines would have been output as one continuous line. For example, if your 2 printf statements looked like:

```
printf (" Programming is fun." );
printf (" And programming in C is even more fun.\n" );
```

Your output would look like:

```
Programming is fun.And programming in C is even more fun.
```

So, as you can see, you must be very careful how you code. Remember: the computer will only do what you tell it to do. Nothing more. Nothing less. :-)

The next program illustrates how you can have more than 1 line of text output, with a single printf statement. I already provided you with an example of this concept when I discussed the "audible bell" above. But, it doesn't hurt to provide additional examples!

PROGRAM 2.3

You can take a look at this program and see how multiple lines of text can be output using a single printf statement. I will provide a different example here so to reinforce this concept. Let's simply modify the code in the previous program so that it outputs 2 lines of text using only a single printf statement. The code would look as follows:

```
#include <stdio.h>
main()
{
   printf ("Programming is fun.\nAnd programming in C is even more
fun.\n" );
   getchar( );
   return 0;
} /* end main */
```

As you can imagine, the output of the above program would look like:

```
Programming is fun.
And programming in C is even more fun.
```

Again, the reason this works is because you can embed *escape sequences* within a **printf** text string.

I added a comment on the last line ***(the end block marker)*** and you should always do that. Also, leaving a blank in front of main is the same as putting **int** there, so we still need the **return 0;** statement.

Please try to create, compile and run the above programs on your own computer. Vary the programs a bit and see what types of output you produce. This is an excellent activity for ensuring that the concepts discussed so far are completely understood. The concepts discussed throughout this course will remain far too abstract if you do not take the time to practice what you are learning.

Creating and Running a Program In C-Free:

1. Run C-Free

2. Select: File > New

3. Type in the above (or copy and paste it in.)

4. Select: Save as > name it something like fun.c (the .c extension is important!) Put it in a convenient folder.

5. Select: Build > Run (or just press F5)

If the program is good it will compile and run in a separate window. If not there will be error messages (that do not always tell you what the error is, but usually the line numbers are helpful)

That is it!

Also, throughout the textbook you will see some QUICK TIPS. Please take the time to read them as they provide some tried-and-true programming advice!

The next set of lecture notes introduces the very important concept of program **variables**. Perhaps you should go get a very strong cup of coffee at this time!