Lesson 1 Getting Started

Writing your first program in C

Example 1. **one.c**

Open up an editor of your choice and type in the following program

```
#include<stdio.h>

int main()
   {
    printf("Welcome to the world of C \n");
    return(0);
   }
```

Save the file as one.c

Be careful to type in the code exactly as you see it, braces not brackets etc. Depending on your C compiler and on whether you are using the command line or an IDE, you can now compile your code.

The .c file which you have just written is “understandable” to the human but not to the computers CPU.

The compiler is the first step in converting your source code (.c) to machine code (binary file). Having compiled your source code successfully you now will have an .obj file or .o file depending on the platform that you are working on. The next step is linking. Linking is necessary for several reasons.

Firstly, your program needs to be combined with several library routines. Secondly, you can link different files, which may have compiled independently. After linking your file(s), you now should have an executable file – this may be “a .exe” or “a.out” file. So the original .c gets compiled to produce an .obj file which when linked produces an .exe file. Keep in mind that the object file and executable file will have different extensions depending on your coding environment. In addition, depending on the setup, the compiler may produce an executable directly from the source code.

The basic structure of a C program

The word **main** is very important and must appear once and only once in every C program. This is the initial point of execution when the program is run. We will see later that this does not have to be the first statement in the program but it must exist as the entry point. C programs can be divided into units called functions. Note that **main()** is a just a function.

A pair of parentheses indicates to the compiler that this is a function. Functions will be discussed later. The word **int** proceeding **main** specifies that the function return an integer on completion.

The curly brackets, called braces, are needed to define the limits of the program itself. The actual program statements must lie between the two braces. Note that braces can also be called delimiters.

The line in the program beginning with “printf” is an example of a statement. A statement in C is terminated with a “;” . Note also that a semicolon terminates the line not the carriage return you type afterwards.

The C compiler pays no attention to the carriage return or indeed “whitespace” characters.

The function **printf()** prints to the screen the argument supplied inside the parentheses. Thus, the string "Welcome to the world of C" will appear on the screen when the program is run.

Finally, the function **main()** returns an integer via the **return()** function.

Printing Numbers

Example 2. **try2.c**

```
#include<stdio.h>
int main()
   {
   /* create a variable to store a num */
     int num;
     num = 7;
     printf("The value of num = %d\n",num);       num = 4;
     printf("The value of num = %d\n",num);       num = 0;
     printf("The value of num = %d\n",num);       num = 10;
     printf("The value of num = %d\n",num);
     return(0);
    }
```

Type in the program try2.c

Compile it and then run it. Have a look at the output on your screen. The first new thing that you encounter is the line containing

int num;

This defines an integer variable named num. The word **int** is a C keyword and cannot be used for anything else by the programmer, for example a variable name. It defines an integer type of variable that can store a whole number within a predefined range of values. The actual range is defined later.

The variable named “num” can be any name that follows the rules for an identifier and is not one of the keywords for C.

The final character on the line, the semi-colon, is the statement terminator as discussed earlier.

Note that even though we have defined a variable, we have not yet assigned a value to it, so it contains an undefined value. Defining and assignment can take place on the same line if required.

Observing the main body of the program, you will notice that there are four statements that assign a value to the variable num, but only one at a time. The first statement assigns the value of 7 to the variable num, and its value is printed out on the next line. Later, the value of 4 is assigned to num, then 0 and finally 10 is assigned to it. It should be clear that num is indeed a variable and can store many different values but only one value at a time.

We use **printf()** statements to print out the variable num. The **printf()** function is the workhorse output statement in C. The character **%** is a special character that signals the output routine to stop copying characters to the output and perform something different, for example output the value of a variable. The **%** sign is used to perform the output of many different types of variables. The character following the **%** sign is **d**. This signals the output routine to get a number and output it to the screed. Note that **%d** is called a format specifier. After the **d**, we find the familiar **\n** (newline), which is a signal to return the video "carriage", and the closing quotation mark.

Comments in C

It is helpful to be able to put comments into source code that are read by humans but are invisible to the compiler. A comment begins with /* and ends with */. Comments should be added where there is a possibility of confusion. Most compilers do not allow the nesting of comments, so the following may be illegal "/* this is not allowed /* because */ my compiler says no */". Check with your compiler.

Good programming practice would include a comment prior to the program with a short introductory description of the program. Comments are very important in any programming language because you will soon forget what you did and why you did it. It will be much easier to modify a well-commented program a year from now than one with few or no comments. You will very quickly develop your own personal style of commenting.

Programming Style

You should try at all times to write a well formatted and easy to follow code. This means that it is reasonable easy to see at a glance what the program actually does. It also allows your team become familiar with the code. Your C compiler ignores all extra spaces and all carriage returns giving you considerable freedom in formatting your program. Indenting and adding spaces is entirely up to you and is a matter of personal taste. It is highly recommended that your coding team document and adhere to an agreed style. Read up on some examples of coding styles.

Programming Exercises

1. Write a program to display your name and age on the monitor.
2. Modify the program above to display your address and phone number on separate lines by adding two additional **printf()** statements.
3. Write a program to print out the result of 3.14x1.96 correct to one significant digit.
4. Find out what the library stdio.h is and the what the puts() function do.
5. Without using the printf() function, write a program to put your name on the screen.
6. Write a program to print out the numbers 2^2,2^4 and 2^5.
7. What are the three basis output functions in C.
8. Write a program to compute 1.0/3.0 and print out the result as an integer and a float.
9. Using a field width specifier and %f, print out 1.0/3.0 with four significant digits.