

Session 2: The C Program Structure

2.1 Introduction

Every programming language has a way of how you can compose program statements for the program be executed by the computer. As such we will look at the general program expected from us as programmers when we a C program.

2.2 The C Program Structure

The following is the general C program structure expected for each that we write in C:

```
#include <stdio.h>

int main()
{
    return 0;
}
```

Figure 1 below has labelled the two major components and its constituent parts:

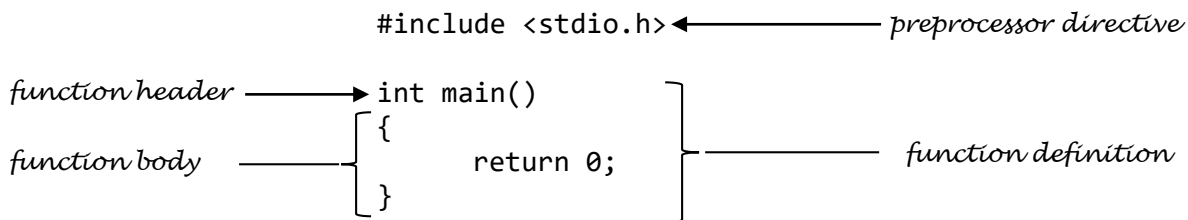


Figure 2-1: The labelled C Program Structure

A C program is composed of two main components, namely;

- i. preprocessor directive
- ii. function definition

The preprocessor directive – is an instruction given to a preprocessor (a major component of the C compiler), commanding it to prepare the code for translation into machine language – in so doing the preprocessor can be instructed to include a standard file, do code substitution among others. We will revisit this line of code later in Session 3.

Function the definition – every C program is written in a C function. The definition is a specification of a C function, in terms of *function header* and *function body*. The function header marks the beginning of a C function and function body defined as a block of statements, which start with an opening curly bracket and ends with a closing curly bracket. A block of statements – is a group of statements that are executed sequential from top to bottom, one after the other.

As of now, we should note that our function header has two key words `int` and `main`, we call this function the `main` function. This is a very important function in C, this where your C program

execution begins. In essence, we cannot execute a C program without main; in other words a complete C program must have **one and only one** main function. Therefore, initially all programs will have one function definition and will be main function.

2.3 Editing, Compiling, Executing and Debugging a C program

Having known how a C program looks, we now ready to edit, compile, execute and debug our first C program. This will be done through an example program as shown in Listing 1 below:

Listing 1: The First C program

```
#include <stdio.h>

int main()
{
    printf("This is my First C Program");
    return 0;
}
```

Editing a C program

We will assume that we are all using the CodeLite as our Integrated Development Environment (IDE). Editing is nothing but typing the program in an editor provided by the IDE in a named file.

1. Having created the workspace and the corresponding project in the workspace, expand the project folder – this will list all the project folders in the project.
2. As a basic project that you created there is a default folder created for us called the src folder, right click the src folder within the project folder and click the Add a New File ... option from the pop menu displayed as shown in Figure 2-2 below:

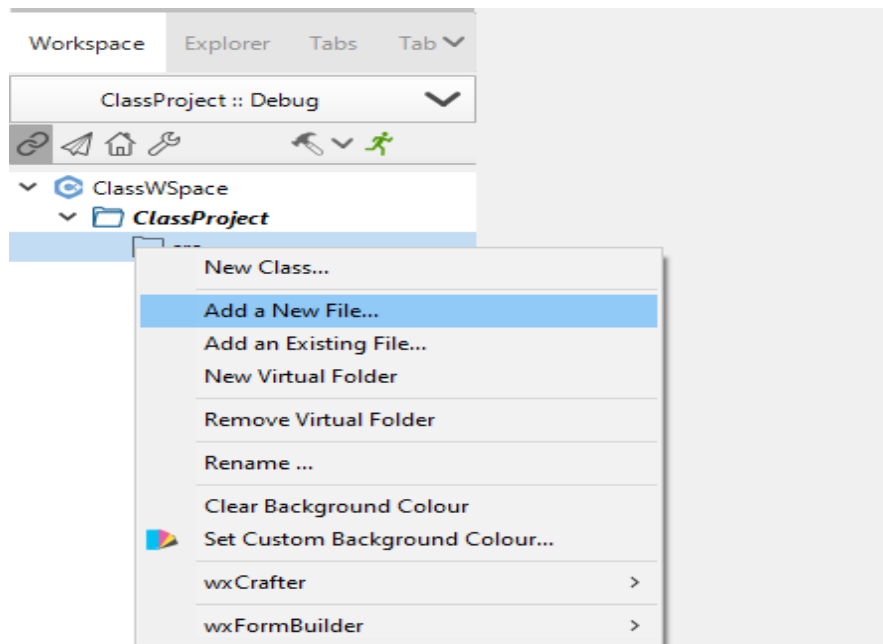


Figure 2-2: Creating a New C program File

This will display the New Item dialogue box where you are required to specify the File Type in the File Type section – this is done by clicking once the File Type C Source File(.c) option. Thereafter type the File name of you C program in the textbox labeled Name:, in our case we will call it First. A complete filled in New Item dialogue box is shown in Figure 2-3 below:

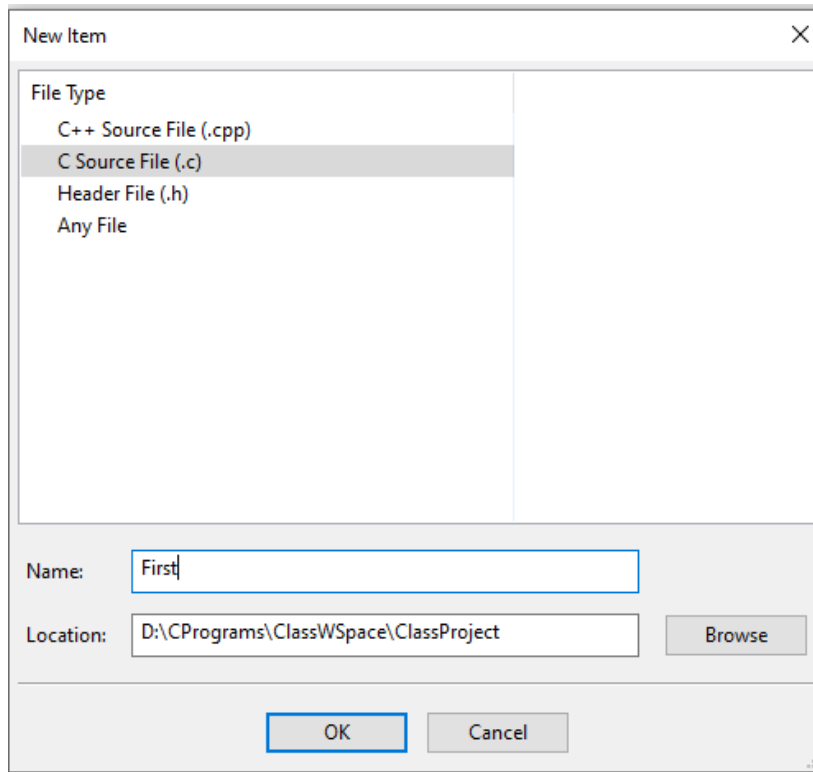


Figure 3: The New Item dialogue box

3. You are now ready to type the code in the Edit window of the First.c file as shown in Figure 2-4 below:

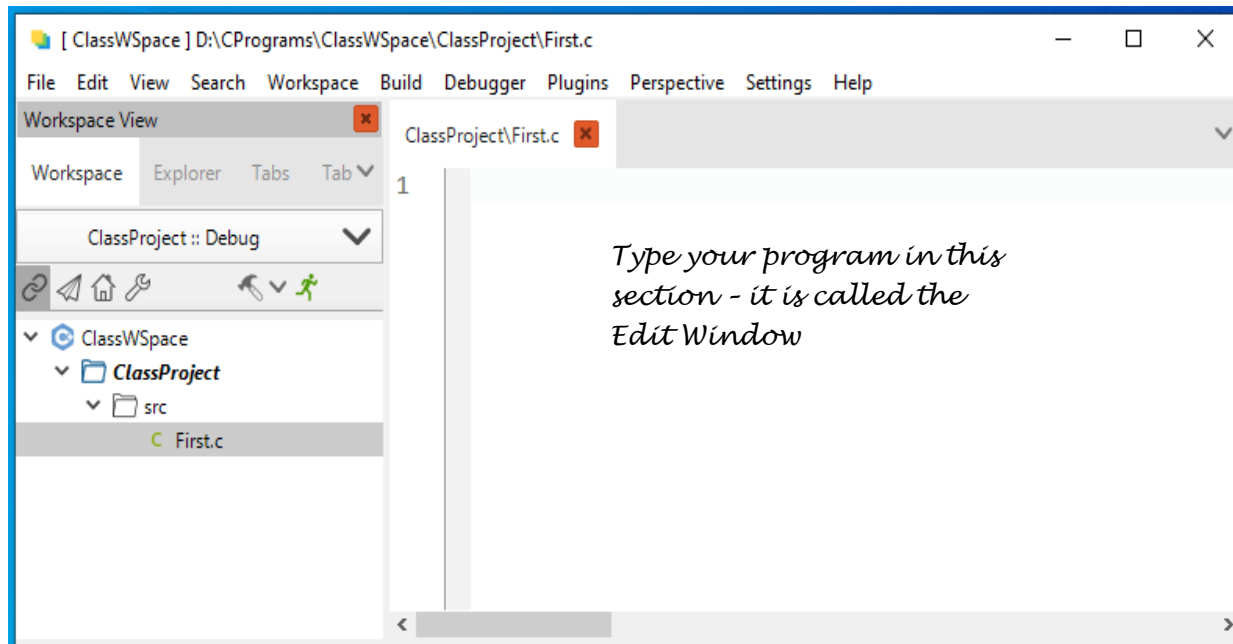


Figure 2-4: The source code file Edit window

After typing the example program shown in Listing 1 above, the Edit window will look as shown in Figure 2-5 below:

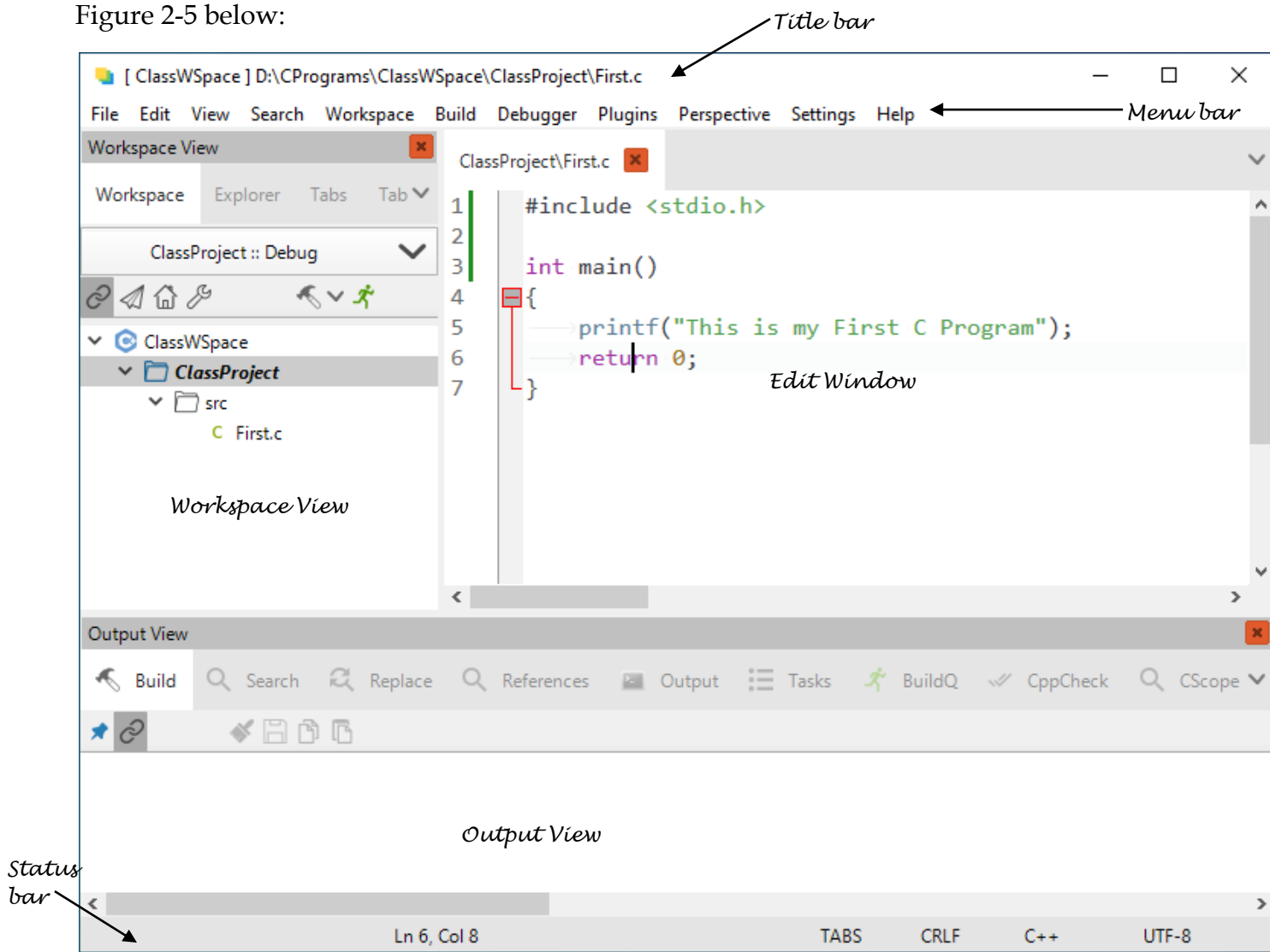
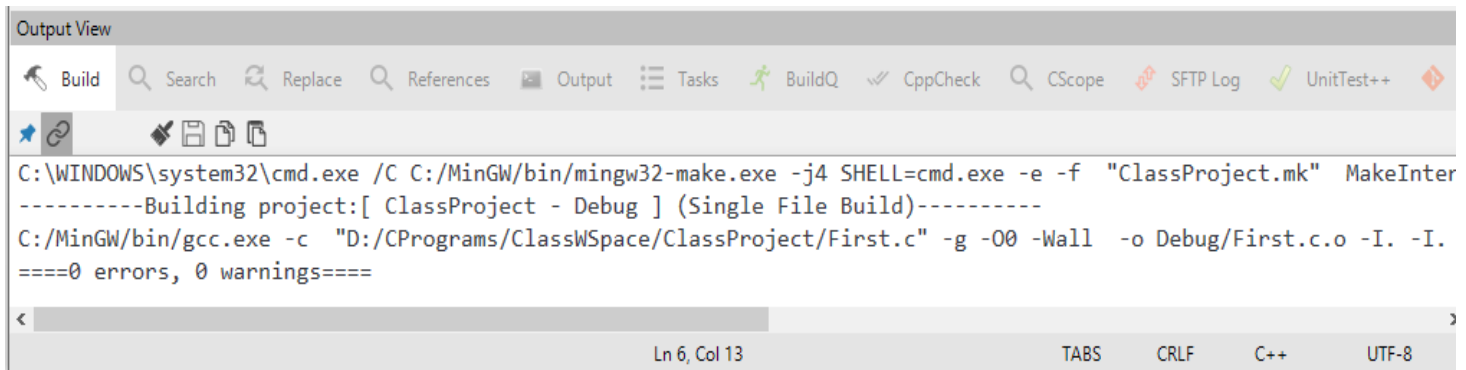


Figure 2-5: Typical View of a C Project, with a typed source code file

Compiling a C program

Compiling a C source code file is very simple from the IDE. Click Build from the menu bar, then Select Compile Current File; this compile the current file and if no errors the output view in the Build tab will display the text as shown in the Figure 2-6 below:

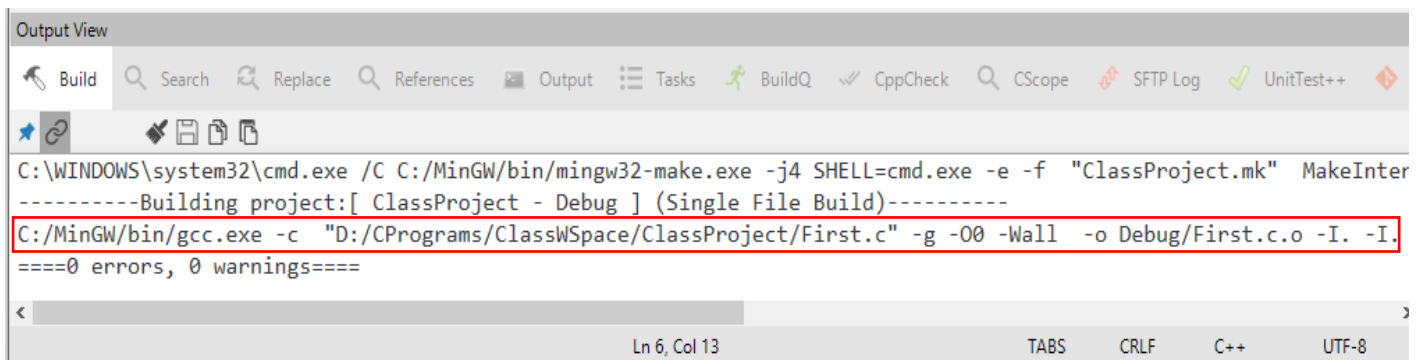


```
C:\WINDOWS\system32\cmd.exe /C C:/MinGW/bin/mingw32-make.exe -j4 SHELL=cmd.exe -e -f "ClassProject.mk" MakeInter
-----Building project:[ ClassProject - Debug ] (Single File Build)-----
C:/MinGW/bin/gcc.exe -c "D:/CPrograms/ClassWSpace/ClassProject/First.c" -g -O0 -Wall -o Debug/First.c.o -I. -I.
====0 errors, 0 warnings====
```

Figure 2-6: The Output View displaying “====0 errors, 0 warnings====” message

In this context the Listing 1 above has no errors and therefore the Output View should display the message “====0 errors, 0 warnings====” in affirmation as expected.

From the Output View window of the IDE, it is easy to note that the gnu gcc compiler, in an instruction just above the message “====0 errors, 0 warnings====”. The instruction we are referring to is labelled and is shown in Figure 2-7 below:



```
C:\WINDOWS\system32\cmd.exe /C C:/MinGW/bin/mingw32-make.exe -j4 SHELL=cmd.exe -e -f "ClassProject.mk" MakeInter
-----Building project:[ ClassProject - Debug ] (Single File Build)-----
C:/MinGW/bin/gcc.exe -c "D:/CPrograms/ClassWSpace/ClassProject/First.c" -g -O0 -Wall -o Debug/First.c.o -I. -I.
====0 errors, 0 warnings====
```

Figure 2-7: The Compilation Instruction that invokes the gnu gcc compiler

Compilation is a process that has an output, in C the compiler generates an object file when there no errors found in the source file. The compilation command has many arguments but we need to take note `-o` argument, which specifies the output file after compilation. This instruction specifies that the output file, which is the object code file is called `First.c.o` and stored in a sub folder `Debug` in the project folder `ClassProject`. The whole path for the project is also specified in the same instruction, so we can easily get the object file in this case as: `D:/CPrograms/ClassWSpace/ClassProject/Debug`. It is very important as a programmer to know the location of your files, as rule please do not use the desktop as part of file storage; please create your folders in a drive that you access and retrieve without a hustle.

Execution (Running) a C program

Using the IDE, executing an error free program is very easy. Select the menu Build, click the Run menu option; this will run the C program and display a console window showing the output of our program as shown Figure 2-8:

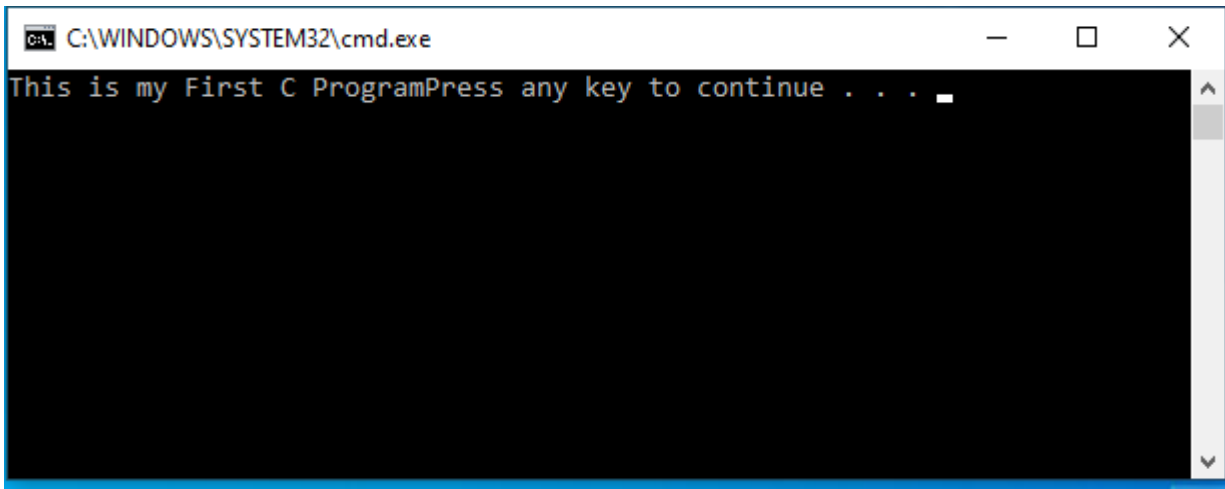


Figure 2-8: The console output window displayed when we run a C program

It displays the constant string “This is my First C Program” as given to the printf function; however, there are some words that are displayed immediately after that, the string “Press any key to continue...”. These words are used by C compiler to pause the running of the program until we press any key as suggested. Please not the word Program and Press are too close by such that we can read it properly. To help distinguish between the pause message and the constant string of the program, we will modify the source code and insert an escape sequence which tell our compiler to the pause message on its own line using carriage return escape sequence (“\n”). We will place this escape sequence together with the constant string in the program “This is my First C Program\n” as shown in Listing 2 below:

Listing 2: The Modified First C program

```
1    #include <stdio.h>
2
3    int main()
4    {
5        printf("This is my First C Program\n");
6        return 0;
7    }
```

Saving, compiling and executing the Listing 2 gives the following output in the console window as shown in Figure 2-9.

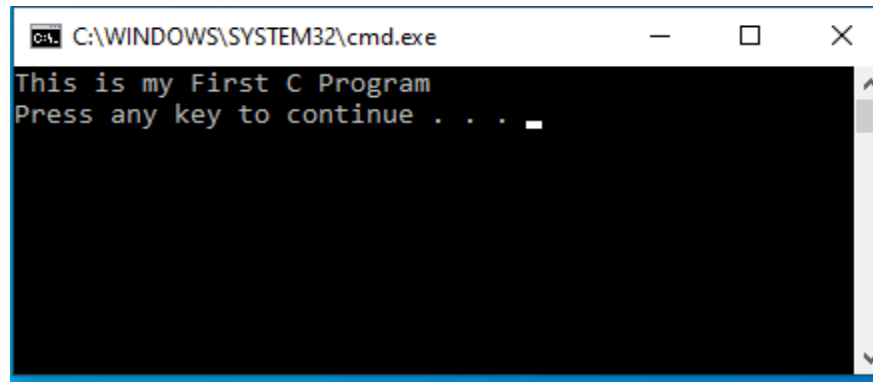


Figure 2-9: The console window for Listing 2

You can see that the pause message, is in a new line own its own; courtesy of the newline (carriage return) escape sequence. An escape sequence represents the characters that we cannot type in program statement. For example, the tab key, the enter key (carriage return key), the bell sound among others. The following are some of the escape sequence characters used

Escape Sequence	Description
\n	Carriage return
\t	Insert a tab space in text at this point
\a	Bell sound
\f	Inserts a form feed in the text at this point

Debugging a C Program

We need to modify the Listing 2 above a bit to see how we can deal with errors in a C program. They say that errors are part of programmer's life, we have to live with them; they were written for us – therefore there is need to learn how to resolve errors in program statements. Remove the semicolon on the program statement labelled line number 5 in Listing 2 to have the following code in Listing 3.

Listing 3: The Debug First C program

```
1    #include <stdio.h>
2
3    int main()
4    {
5        printf("This is my First C Program\n")
6        return 0;
7    }
```

On compiling Listing 3, the program has the following error as shown in the Output View under Build tab in Figure 2-10:

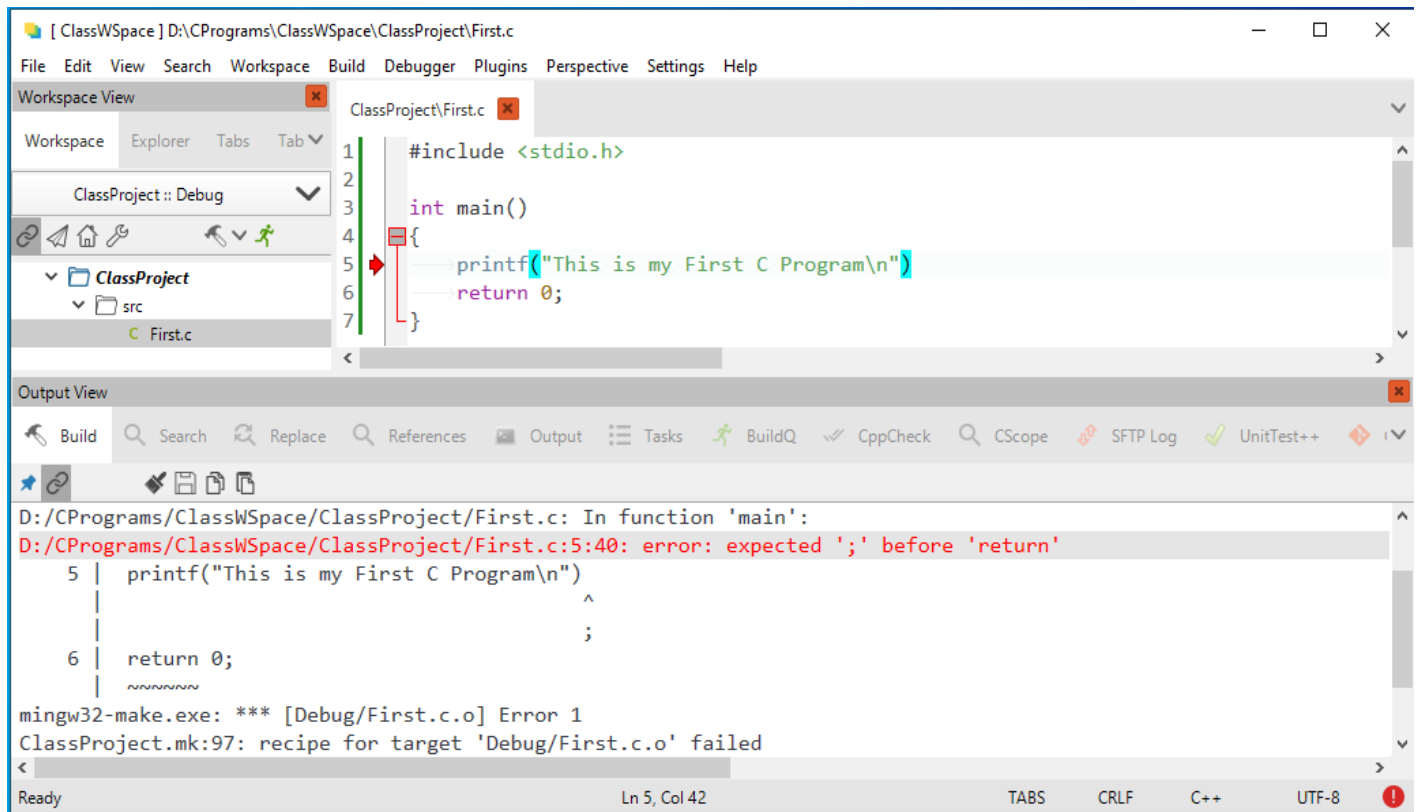


Figure 2-10: Resolving Errors in a C program

It is important to know how to read errors so that you are able to solve them. The compiler is a good tool and the IDE packages errors in more user friendly manner. Like for our case in Figure 2-10 above, we are told in the Output View Build tab, the error line text is highlighted in red reads as follows:

D:/CPrograms/ClassWSpace/ClassProject/First.c:5:40: error: expected ';' before 'return'

This text is self-explanatory, first it gives the path of the program First.c before the first colon (:), then the line number that has a problem which in this case is line number 5 and column 40. This is followed by the word error and then it gives the error expected ';' before 'return'. Simply telling us that it expects a semicolon at the end of line number 5 but where just before the keyword return. Below the error text the line that follows tells us where the semicolon is expected just after the closing parenthesis ')'. You realize errors are easy to sort only if you read and understand what is required of you. If you click on the error text, a block arrow will place on the line number 5, and when you place the mouse pointer on the block arrow it will show a pop text of error message.

The advice on errors as of now is that you need to read and understand; follow the suggestions given in the Output View Build tab and it will be easy to solve those errors. To help you also take care of many errors, always start from the first error solve it then compile before solving the other errors; the first error might be having a relation to the other errors that you in the error list. Always sort the first error before dealing with the rest of errors after compilation. The error which is just presented is a typical syntax error – obtained as we compile our source code.

2.5 Output Statement

In section 2.4, we introduced a way of how to read errors – just familiarize ourselves with basic syntax errors. This is also, was in a way to introduce the concept of a C program statement; a statement ends with a semicolon ‘;’. A qualified C statement ends with a semicolon ‘;’, just like in a natural language a sentence in English ends with a full stop ‘.’. Therefore, that is why the compiler was complaining about a semicolon ‘;’ expected at the end of the statement in line number 5 of Listing 3.

Line number 5 in Listing 3, presents an output statement; it is a statement that is used to display, store or print output data – in form a constant string, picture, file among others. This specific output statement displays the constant string on Visual Display Unit commonly called the screen, it uses the C function called `printf()`. The `printf()` function is defined in the header/library file called `stdio.h`; that is why we instruct the preprocessor in the include preprocessor directive to include the header file `stdio.h`.

Exercise

1. Modify the example program to display another constant string “Welcome to CSE 4101 Introduction to Programming”.
2. Create another program which will display your address as follows:

Katana Kahindi Jamba
Marereni Primary School
P.O. Box 56789 00420
Gogoni
Kenya

Hint: use the tab space escape sequence in the line of text that has the post code.