Index

Please do not use this page as an indication of content. This book contains dozens of programs which are not mentioned here.

Chapter 1 – Introduction

In order to learn any language, it is necessary that the person learning must firstly dominate the language alphabet. Then the person can begin to formulate words, then complete sentences according to the language syntax. We may understand that the syntax are the rules which we have to follow so that a sentence may be understood by a person listening to or reading the sentence. Even so, we can verify every day the fact that some people ignore the syntax of the English language. Nevertheless we can usually work out what these people are saying. However, when we're writing a computer program, in this case C++ the Compiler doesn't contain inbuilt artificial intelligence, nor other methods which help de-cypher out of syntax instructions and will either do something daft, or emit an error message. So in order to avoid problems with the Compiler, the syntax has to be followed exactly. This makes sense because we want our written program to work exactly as planned, every time it is executed. That said, a good programmer must think logically and if you understand that if A = B and A = C then B = C and that's good enough to start. Some important questions might arise, like "What is a Compiler and what does it do?" and "What is a Pre-processor?". "What does an Assembler do?". "What is a Source Program? What is a Linker? All these questions will be answered now.

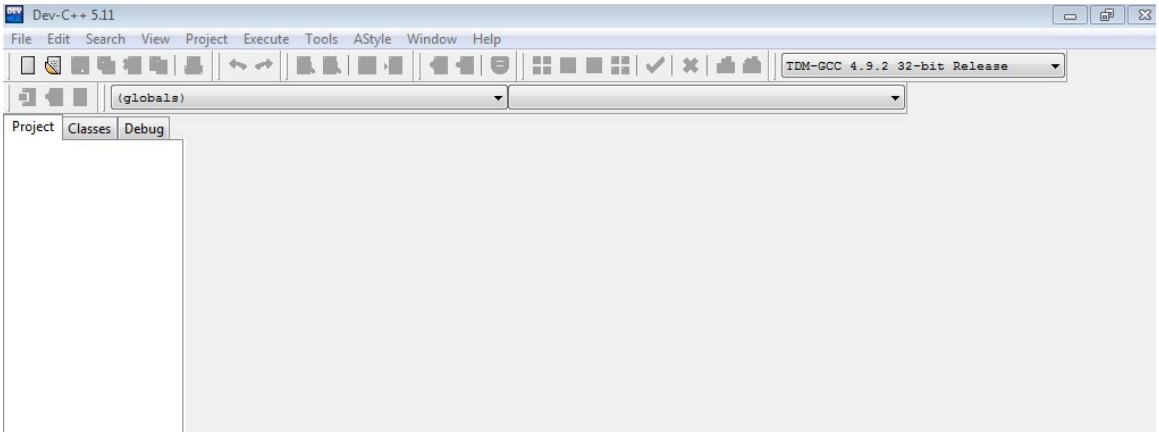
As we shall see, a Compiler has many parts and does all the following tasks:
1 A Pre-processor processes the Macros, Instructions and Conditional Compilation as well as the Header files of type Include;
2 A Compiler gathers the results generated by the Pre-processor and your Source Code and generates the assembly of the Source Code;
3 An Assembler gathers the code generated by the Compiler and produces an Assembly List including displacements. The output from the Assembler is put into the Object Code File;
4 A Linker gathers the Object Code File as input and combines it with the Header Files. The result is the executable file which has the .exe extension.

Note that the Compiler saves two files, one with the .cpp extention, which is your Source code, the other with the .exe extention, which is your executable program.

So, all that said, let's grab a suitable development system, the Dev C 5.11 Integrated Development Environment at the link below:

https://sourceforge.net/projects/orwelldevcpp/?source=directory

Download the setup file and save it to your download folder. Go to the folder and click the setup file. Just follow the instructions to install the system. You'll then see a link icon which is a blue square with DEV written in it on your desktop space. Click the icon. The program may say that it hasn't detected a path to the Library but the system does give you the option to have one created. Open the system. You will be presented with the DevC++ starting screen as seen below.



Have a look around clicking on Tools tab, Compiler options, see which Assembler is installed. Make sure it's the 32 bit option if your computer is 32 bits.

Now click on File tab and New, then Source File. A workspace appears to the right of the Project space. The new space is where you'll write all your Source Code. Click the park (-) tab at the top right. Click on Start icon on bottom left. Click on Documents and create a New Folder. Name it My C++ Progs or something you might like better and Ok. Click the DEV icon to bring up your screen again. At the moment your program hasn't a name. By default it has been named Untitled1. Click File, Save as...a suitable name into your new C++ progs folder and OK. Now the name appears where Untitled1 was.

Now we can write our first C++ program. Every program must have a library name, the Header. Headers tell the Compiler how to process certain parts of your source code and there are many libraries, as we shall see. For now, though, we'll declare the inclusion of a library called iostream. This library deals with Output and Input, specifically cout and cin.

Chapter 2 – The First Steps

At the top left of your Source Code workspace, type the following line.

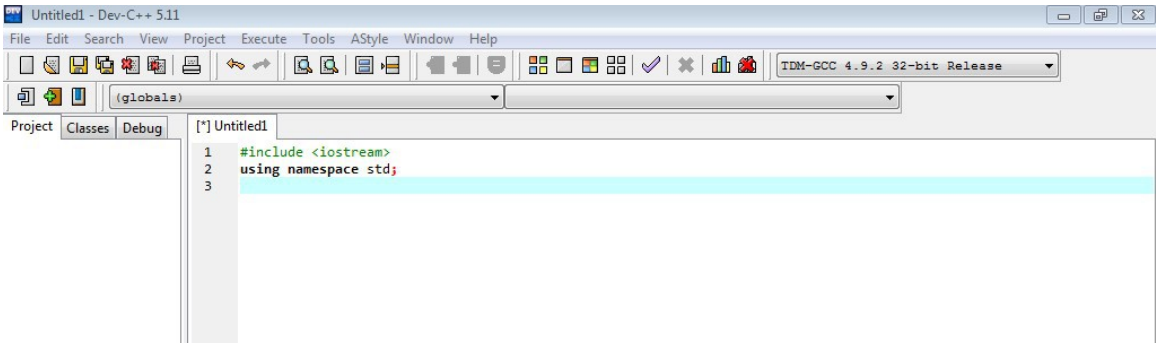
#include <iostream>

This is a directive to the Pre-processor to include a library called <iostream> whose code should be added to our Source Code program, before creating the executable file. C++ already has several of these coded programs which we shall include depending upon the work our program should do. Basically, iostream tells the compiler how the program should send characters to our screen, as well as those originating from our keyboard by executing the functions cout and cin. The include directive is called the Header and every program shall include at least one Header File.
Going back to our Source Code workspace, under the Header, type:

using namespace std;

Note the semi colon at the end of the line. This tells the compiler that we've terminated the instruction and is part of the C++ syntax. We'll be using namespace std; in all our programs and its significance is explained in detail in chapter 15. Briefly, it tells the Compiler to use a group of functions which are part of the Standard library. For now, let's just tuck it in under the Header.

Now, your workspace should look like the picture below:



Notice that each line of source code is allocated a line number, which shows lines 1 and 2 to the left of the code at the moment. Line numbers make it easier to debug our code, as we'll see later on.

Our first program's purpose will be to send text to the screen. It seems simple, but we'll learn a whole lot about C++ while doing it.

Under our second line of code, type:

```
int main ()
{
```

We've just added a function called “main ()”. This is always the first function to be executed and every program will contain a main function. The brackets () following main indicate that this is a function, and that the brackets may contain parameters.

The type "int" before main indicates that the main function can return data.
Now let's look at the curly left bracket {. This tells the compiler that the main function starts there, and all code up to the matching right bracket } should be executed. Now, complete your source code just like the one below, so your program should look like this:

```
#include <iostream>
using namespace std;
int main()
{
cout << "Greetings Earthlings, here goes my first C++ program"
<< endl;
return 0;
}
```

Here, C++ sends all the characters within quotation marks to the function cout, which sends the text to the screen. The flow is from right to left in accordance with the << operator. Notice the inclusion of << endl;. This tells the compiler to send the cursor to the beginning of a new line, just like pressing the Enter key, so we know that we may use multiple instructions on the same line as cout by separating each instruction with the << operator.

Now we can execute our first program. Look for the Execute tab and click it. The drop down dialogue contains a few tasks, the first being "Compile", the second "Run" if we wish to compile and execute our program separately, we can click these in turn, or we can use the shortcut keys F9 and F10. We also have the option to compile and run our program in one operation, or use the F11 key.

Notice that your anti virus program may not allow you to execute .exe programs and may even corrupt your Dev-C++ installation in the worst case. Best to temporarily switch the anti virus off before running the program.

Once the program has run, C++ stops at the right curly bracket, and it does this so that we may see what cout has sent to the screen. If the program were to run straight through the } curly bracket we wouldn't ever see the text printed on the screen because the execution is too quick.

Once you're happy with the running of your program, click File then Save as...the name you chose for your first program, to the folder you created for all your C++ programs.
Note especially upon saving the program that if you change the program at any time, C++ tells you that the program has changed and gives you the options to reload from disk or not. Do NOT click the Yes button, click the NO button because if you click yes, you'll lose your .cpp source program.

Chapter 3 – Adding comments

Our first program is quite simple, but when we develop larger and more complicated ones, it's a really good idea to add comments about what our lines of code are supposed to do individually. Even a piece of code written the night before can baffle our intellect to the point we ask ourselves next morning, "now for what purpose did I write this line of code?" We'd know had we put in a comment after the offending line. We tell the compiler that "this line or these lines of text are comments" and they shouldn't be processed.

Example:

```
10 instruction; // "This is a comment"
11 // instruction; This instruction will not be executed
12 instruction; /*Start of block comment
13  Compiler knows this line is part of block
14 */End of block comment
```

Under the source code space you'll see the tab called Compile log. Click it if it's not selected already. The system uses the Compile log to show all compilation errors, as well as the time taken to compile the program and the correct running of the program.

Congratulations, you've written your first program using the C++ compiler. Admire your work and then play around with it and see which type of error messages the compiler prints out. Ok now, we can get off Twitter and Facebook showing them your C++ program. But do you really remember all the little details which have made your program work? Hmmm. Here's a little test, just to find out.

A small test. Test 1:

1) Which is the one function that all C++ programs must contain?
a) include
b) main()
c)`return
2) What function do we use to direct characters to the screen?
a) send;
b) using;
c) cout <<;
d) None of the above;
3) To tell the compiler where to begin main(), we use?
a) [
b) >>
c) }
d) {
4) What kind of punctuation do we use to end a line of code?
a);
b):
c),
d).

5) Which header file do we include to use cout?
a) None. The system automatically does it for us.
b) <stringstream>
c) <iostream>
d) using namespace std;
6) Which of these comments is correct?
a) */ comment */
b) / comment
c) comment //
d) //comment

7) Write a program which sends your first name to the screen. Did it compile correctly? Did it run without errors? If so, congratulations, If not, look at your source code for a missing ; at the end of the line, or { at the beginning of main(). Anyway, the compiler will help by highlighting the line of code where the error occurred.

Answers:

Test1:
1)b; 2) c; 3) d; 4) a; 5) c; 6) d;

Chapter 4 – Interactive programming

So far we've gained the knowledge to elaborate a program which sends characters to the screen, which is great. However, it's a one way program and we'd soon fall asleep just looking at it. We want to soar to greater heights, but we must firstly be aware that input data like characters, numbers and text must be stored in the program, more specifically, in memory. We use for this purpose the so-called variables, and there are several types of variables, which must be named, declared and initialised by our program before they can be used. The following list shows all the types including limits of variables we may use:

"short int" may be any number between –32.768 and +32.767;
"signed int" or just "int" may be any number between ---2.147.483.648 and +2.147.483.647;
"unsigned int" may be any number between 0 and 4.294.967.295;
"unsigned short" may be any number between 0 and 65.535;
"long" may be any number between –2.147.483.648 and + 2.147.483.647;
"unsigned long" may be any number between 0 and 4.294.967.295;
"float" may be any floating point number between 3,4E+/-38 in scientific notation, equivalent to $3,4^{+/-38}$
"double" may be any floating point number between 1,7E+/-308;
"char" a character noted in the ASCII table;
"string" a sequence of characters and spaces;
C++ automatically recognizes all variables with the exception of "string" which requires the inclusion of the header called <cstring>. This library shows the compiler how to process this type of data. Following are a few variable declarations by name and type:

```
int number; /*We've declared a variable called number which
accepts numbers between --2.147.483.648 and  +2.147.483.647;
*/
```

```
string name; /*We declared a string type variable called name
                which accepts a sequence of characters
                including spaces*/
char characters;/*We declared a char type variable named
            characters*/
cout << int ('a character'); // Prints the ASCII code of the character on the screen.
```

The variable char holds only one character. If you wish to store more than one character you'll have to use a matrix of char type, which we'll learn all about later on. If we declare and initialise a char type variable, for example char character = “a” we may write character = '\0' which is the NULL character to delete the “a”.

Now let's get down to our first interactive program without further delay. Firstly we need to know that the black square which appeared when we ran our first program is called the Disk Operating System (DOS) Window. Windows uses the DOS for many purposes, one of them being reading input and writing output.

Click on the File tab, Close option, to clear the source code workspace. Click File again, New and Source file. Click File, Save as ... give our interactive program a suitable name and save it to your C++ programs folder.
What we want our program to do is to ask a user to type in their name and age. These will be printed to the screen in a friendly way.

You'll see your programming workspace which is now blank, just waiting for your new program. Click the top left corner and type:

```
# include<iostream>
# include<cstring>//Use cstring header to store a sequence
using namespace std;//Mustn't forget the semi colon
int main()//Here is the main function
{//The main function starts here
int age;//Declare a type int variable called age. (Only numbers!)
string name;//Declare a type string variable called name
cout << “Please type in your first name.” << endl;
cin >> name;/*We use the cin function which tells the compiler
                    to store the keyboard input in name*/
cout << “Hello “ << name << “ how old are you?” << endl;
cin >> age;//cin function expects a number of type int here
cout << age << “ is a good age to be.”;
return 0;
}//The main function ends here
```

Just a major warning here. Word processors may add invisible special characters in the lines of the above program resulting in various compiler errors, so please do not copy and paste from here. Use the Dev-C++ workspace to type in the programs.

Notice that I've taken the liberty to comment the program, just to keep us updated on what each line of code does. Note too that upon running the program, you must end your input with the Enter key which tells the compiler when the input data has ended.

Now let's look at the program. Notice that we've included a new library called <cstring> which we'll be using whenever we want to input strings of characters from the keyboard. After the main function opening brace, the curly one, { , we've declared two variables, the first of type int named age, which only accepts numbers (see the table of variable limits), the second a variable of type string, called name. We should always declare variables with meaningful names that describe the type of data they will be storing. It helps humans understand what the program is doing. The variable declarations made here mean that they may only be accessed from within the main function. We could have declared the variables before the main function, like, after namespace and the program would still work perfectly. However, the variables declared outside a function, let's call it free space could be accessed by any function, which in most cases is not a good thing to do, as we'll see later. The first cout << sent a request for input from the user, and will wait for an answer through cin >> which tells the compiler to store the input in the variable of type string, called name. The Enter key signals the end of character input. Note too that the direction identifier associated with cin >> points to the variable where the input will be stored.

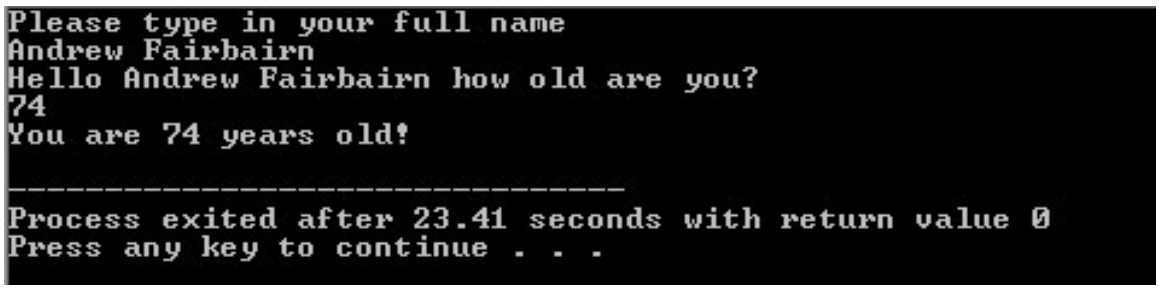
You might have noticed that the cin function has some pretty cramping issues. For instance, it accepts input characters only up to the point it sees a space character, then simply truncates the rest of the input, meaning that it will not be stored. That is the reason we asked the user to input only their first name. So, now we know this fact, we can use a sub-function of cin called getline to substitute cin in our interactive program which now can store a sequence of characters as well as spaces in the variable type string.

The syntax for getline is: getline (cin, variable type string);

getline is a function that uses cin, the second parameter is a variable, each one separated by a comma. You might be asking exactly what are functions? Functions are explained in depth in chapter 17, but for now we can say that we'll be using two types of functions. The first was written by some great guy or girl to help us all write good code, like our header files for instance, while the second type will be written by us and used in our programs to perform all sorts of tasks. Now open a new workspace and type:

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
string full_name;
int age;
cout << "Please type in your full name: " << endl;
getline (cin, full_name);
cout <<  “Hello " << full_name << “ how old are you? “ << endl;
cin >> age;
cout <<  “ You are “ << age << “ years old!” << endl;
return 0;
}
```

Here is the screenshot of the DOS window showing what the executed program does...



In our new program we do have a novelty, look at the string variable declaration. Our compiler won't let us use spaces, but will accept characters like underscores and even a capital letter. Here we used full_name for a meaningful variable name. You can also see that our source program uses two input functions, string for the names and cin for age, and it works perfectly. All said, there is a way of using cin to input two or more sequences of characters, in a process called concatenation by using the + sign.

In your source code workspace, type in the following program.

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
int age;
string first_name, surname, full_name;
cout << “Type in your first name” << endl;
cin >> first_name;
cout << “Type in your surname ” << endl;
cin >> surname;
full_name = first_name + “ ” + surname;
cout << “Type in your age ” << endl;
cin >> age;
cout << “Your full name is ” << full_name << “ and you are “ <<
age << “ years old!”;
return 0;
}
```

Notice especially the declaration of the string type variable. The syntax allows us to declare multiple variables of the same type, each separated by comas in a single line. By doing this, we economize on program space as well as reducing the amount of time our compiler has to work.
The line where we initialise our full_name variable our program concatenates first_name, a space character and surname all in one line and stores it in full_name which is then printed to the screen by our 4th use of cout. Probably we'll rarely have to use this method, but we might have to use it at some time, so there it is.