Lab 0 – Your first C++ program

There are several challenging concepts to master in CS 135. However, before you can learn about loops, functions, and arrays you need to learn how to navigate and use the basic environment of Visual C++. The following exercise is intended to familiarize you with basic navigation of this environment as well as to be used as a reference for your first few programs.

Step1. Open Visual .NET

Step2. Click on "New project".

Step 3. On the left panel, under Visual C++ Projects select "Win32". And from the right panel choose "Win32 Console Project". Then enter a name for the project. Then click on Browse button and select "H:" drive for the Location to store the project. Press the "OK" button, this brings up a wizard.

Step 4. On the left Blue panel click on "Application Settings". Application type should be "Console Application". Under "Additional options" choose " Empty Project". Click "Finish" button. This should bring up a Solution explorer on the left panel with your given project name. Now we need to open a new C++ file.

Step 5. From the top menu bar click on Project. And select "Add New Item". This should bring up a new window. On the left under "Catagories" select "Visual C++". On the right under "Templates" select "C++ file (.cpp)". Enter an appropriate file name. Then click on Browse button and select "H:" drive for the Location to store this file. (*Note: This should be the exact same location as your project is saved at in Step 5 above).*

Step 6. In subsequent assignments the source code you will enter will be different, but for this assignment you will enter the following code EXACTLY as it appears here … not even a semicolon should be different.

```
#include <iostream>

using namespace std;

void main ()
{
     //Variable declarations.
     int x, y, sum;

     cout << "Enter two numbers with space or enter in between: \n";
     //Prompt the user to input two numbers.
     cin >> x >> y;

     //Arithmetic operation '+'.
     sum = x + y;

     //Basic output to the screen.
     cout << "The sum of " << x << " + " << y << " is " << sum << endl;
     cout << x << " - " << y << " = " << x - y << endl;
}
```

Step7. Save the file by pressing Ctrl+S on the keyboard. Now that you have completed entering your source code and saving your file you must '**compile**' your program. Select 'Build' from the menu bar, then select 'Compile' from the pull down menu.

Step8. The box at the bottom of the screen should read 'Build 1 succeeded 0 failed 0 skipped' If it says you have errors, you have entered some of the code incorrectly. In that case you must find the errors and correct them before proceeding.

Step9. Once you have compiled successfully you must '**link**' or build your program. To do this you again select 'Build' from the menu bar but this time select 'build' from the pull down menu. Now the bottom of your screen should say 'linking ….' and then '0 errors, 0 warnings.'

Step10. At this point you have an '**executable file**' and you are ready to '**run**' your program. Click "Debug" then "Start Without Debugging " or press Ctrl + F5. A black DOS window should appear … it should read:

```
Enter two numbers with space or enter in between:
3 4
The sum of 3 + 4 = 7
3 – 4 = -1
Press any key to continue
```

There are several things to consider about this simple program. First lets talk about the structure of a program and the process of building an executable. The text that you typed in is called the '**source code**.' Source code is code written in a high level language like C++. We call it a high level language because it is similar to English and can be easily read and written by people.

When you '**compiled**' you let a program called a compiler check your program to see if it was '**syntactically correct**.' To say a program is syntactically correct means that it follows the rules of C++, much in the same way a properly constructed sentence follows the rules of the English language. It is important to note the difference between being logically correct and being syntactically correct. Consider the English language sentence that follows: "I drove to work in my car" This sentence makes sense .. it is logical, but since it doesn't end with a period it is not syntactically correct. In the same way the C++ statement "cout << "Hello, I'm missing a semicolon" << endl" makes sense to you, but a compiler won't understand it because it doesn't end with a semicolon. Conversely, a sentence can be syntactically correct but really not make sense. Consider the following sentence "My cat drove a fish to the moon yesterday." This sentence has proper subject verb agreement and ends with a period but obviously makes no sense. We say this sentence is syntactically correct but not '**logically correct**.' You will see plenty of examples of logical errors in the C++ code you write throughout the semester. We will talk more about errors in later labs.

The second step, ‘build’ linked your program to the libraries it needed to run. The very first line of the program ‘***#include<iostream>***’ gave your program access to C++ code other people have already written that lets you use the ‘***cout***’ command.
The product of all this is an ‘**executable file**.’ This is a program just like Microsoft Word or Adobe Acrobat (though much simpler). An executable file is a piece of software that actually runs on your computer.

Now we can talk about the individual parts of the program. We already discussed the first line ‘***#include<iostream>***’ This line is called a ‘**preprocessor directive**’.
The next structure we want to consider is ‘**main**’:

```
void main ( )
{
        ….
        ….
        ….
}
```

Every program you write will have a ‘**main**.’ Don’t worry too much about the keyword ‘***void***’ right now, you will learn more about this when we discuss functions later in the course. The important thing to note for now is that you must have a main and your code (excluding preprocessor directives) should go between the curly braces of the main.

Now we’ll consider the ‘***cout***’ and the ‘**insertion operator’.** We use ***cout*** to print to the screen. We can use ***cout*** to print ‘**text**’, ‘**variables**’, or ‘**literals**’. When we print text we must enclose it in double quotes. When we want to print the contents of a variable we simply put in the variable name in the statement. If we want to print a literal we put the literal in the statement. The ***cout*** must be separated from the first component by an insertion operator(<<), and every component must be separated from every other component by an insertion operator.

Another thing to point out is the use of the ‘***endl***.’ Notice that to move to the next line on the screen we must print an ‘**end of line**’ character … notice how the ***endl*** is used in the program. It is just like pressing the “Enter” key while typing on a keyboard to generate a carriage return. Finally notice how the ‘***cin***’ works. How is it similar to ‘***cout***’? How is it different? We call the operator we use with ‘***cout***’ the insertion operator, what do you think we call the operator we use with ‘***cin***’?

<u>Keywords to remember:</u>
Source code, compile, executable file, run, syntactically correct, logically correct, link.