Week 3 - Introductory Programming Lab 1

Exercise 0: Using the Microsoft Visual Studio Integrated Development Environment (IDE)

Although it is perfectly possible to write a text file containing your C programs in a text editor of your choice; invoke the various compilers, linkers and loaders that you need in order to turn your C into machine code from the computer command line; and then run the program from that command line, most programmers use an *Integrated Development Environment (IDE)* to write anything other than the simplest programs. Such an IDE usually contains a text editor, tools to sort through the piles of program files you have written, a method of easily flagging errors and quickly getting you to the lines of your program that caused those errors, and an easy way to see the values of your variables as you step through your programs.

Because IDEs are designed for professional programmers, they have a profusion of parameters and options, and sometimes seem confusing to beginners. However, you will very quickly find yourself able to ignore all the options that you don't need, and just concentrate on writing, compiling and testing code. IP will use Microsoft Visual Studio. It is invoked from the Windows menu from your *Virtual Machine*:

Fig 1: How to Open Microsoft Visual Studio

Once the IDE starts, choose to use the C++ environment and create a new empty C++ project which will contain either a single C program, or a set of files making up a more complex C

program (*C++ is a superset of the C programming language that allows you to program in object oriented fashion, and do additional cool stuff – but the C++ compilers invoked by the IDE will compile and run C programs just fine*). You will, of course, have to name your project – "MyLab1" would be fine, although you can be more descriptive if you like – and save the project in a directory on your home drive so that you can find it in future.

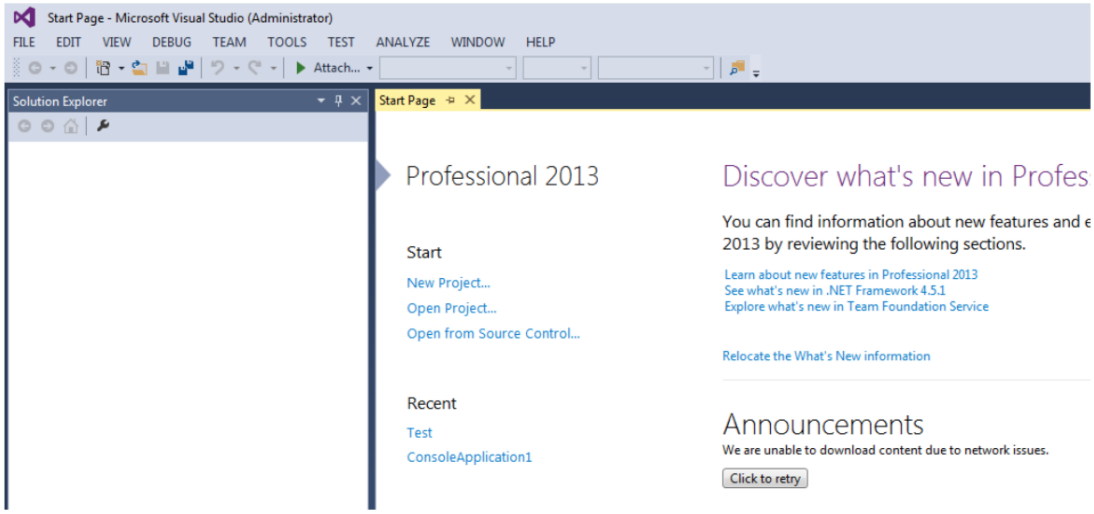

Fig 2: How to create a New project

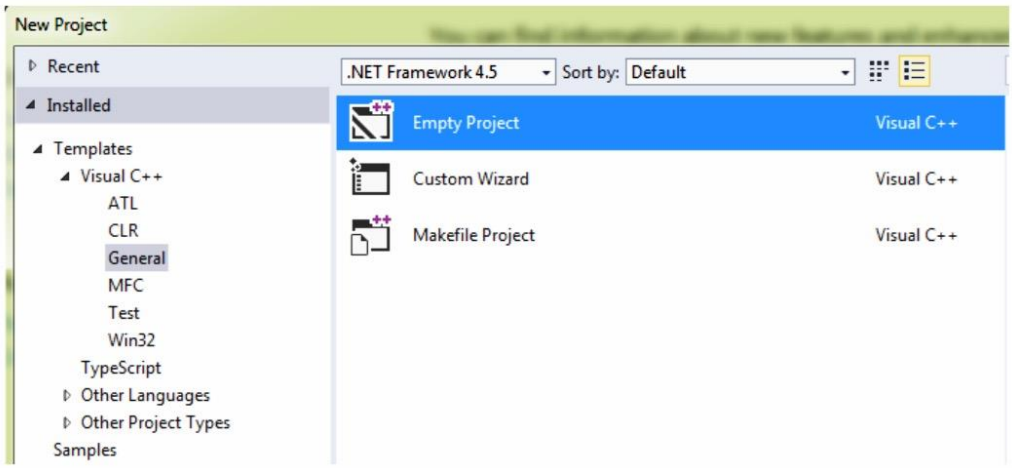

Fig 3: How to choose an empty C++ or C project

The IDE will show a set of windows. To one side are the folders that will contain the files that make up your project (the *Solution Explorer*). Along the bottom is an output window, which will show you the results of your compilation attempts (telling you whether your program has compiled properly or flagging any errors). The large space will contain your actual code, but it's blank because you haven't written anything yet. You want to make a new *source* file, so right-click on the "*Source Files*" folder in the *Solution Explorer* window and **Add** a New **Item...**, and make that new item a C++ file. You can start off by calling this new file **Main.cpp**.

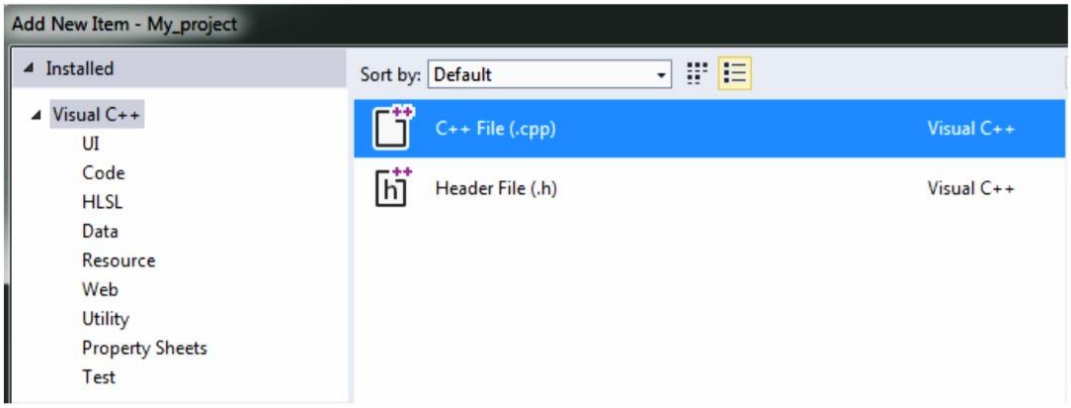

Fig 4: How to create a new file

OK, now try typing in your C code into this new file.

```
#include <stdio.h>

int main(){
    printf("Hello, World!\n");
}
```

You will find that the text editor, knowing that you are writing in C, will try to help you with your choice of **header files**, and colour the code so that you can be more confident that it's correct. Reserved words such as `#include` and `int` will be in blue, and strings and filenames such as `"Hello World\n"` and `<stdio.h>` will be in red. You can save your text by right clicking on the Main.cpp tab at the top of this window.

Now you need to compile your code (and compile any other program files that you might have written, as well as link them all together). You do this most simply by going to the **Build menu**, and invoking **Build Solution**. This does all the various bits of compiling that are needed, so that you don't have to think about which files have already been compiled, and which still need compiled. Very useful if you are dealing with a complicated project!

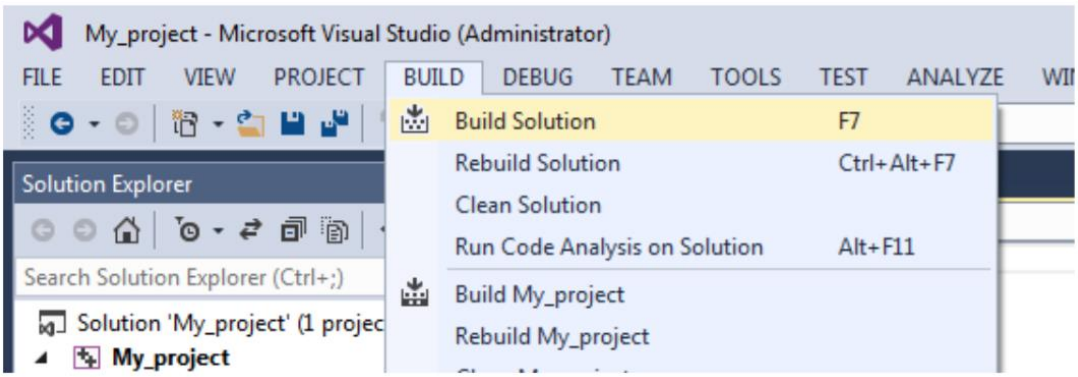

Fig 5: Compiling your program (and other stuff if needed)

In the Output window at the bottom left the results of this compilation attempt are shown. In my copy it contained the line, suggesting success.

```
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
```

Then tried taking out the final close–the quote after `"Hello World!"` to give the following (if you don't see it, try the **View** menu, opening up the Errors window):

Fig 6: Compilation errors

Double clicking on one of these errors takes me to the line of my program that produced the error (which is very useful for long programs). Weirdly, the first couple of errors are *newline in constant* and *syntax error: missing ')'* which might not immediately suggest a missing close quote, but without a close quote the compiler just thinks that the last ')' and ';' in the **prinf** line are more text, it doesn't spot that there has been an error until it gets to the end of the line and thinks "uh?". This IDE also has some additional tools to guess the source of errors, and error 4, *IntelliSense: missing close quote*, does tell you exactly what went wrong. One of the joys of programming is trying to work out where the bugs are in a program, based on a **cryptic** set of compiler errors.

Once you are familiarised with Microsoft's VC++ IDE, you will see that syntax errors in your source code will be automatically underlined in red tilde, almost in real time, just like you have a spelling error when typing in MS Word. Hover your mouse over the underlined, and a tool tip appears to show you what error might have occurred.

Fix your version of the code so that there are no compilation errors. If you have problems, talk to the nearest lab demonstrator/Graduate Teaching Assistance (GTA).

Now we want to run the code, and the three classic ways are to hit **the green arrow** '*Local Windows Debugger*' button in the toolbar, or use the **Debug** => **Start Debugging** Menu item, or to hit the **F5 function key**.

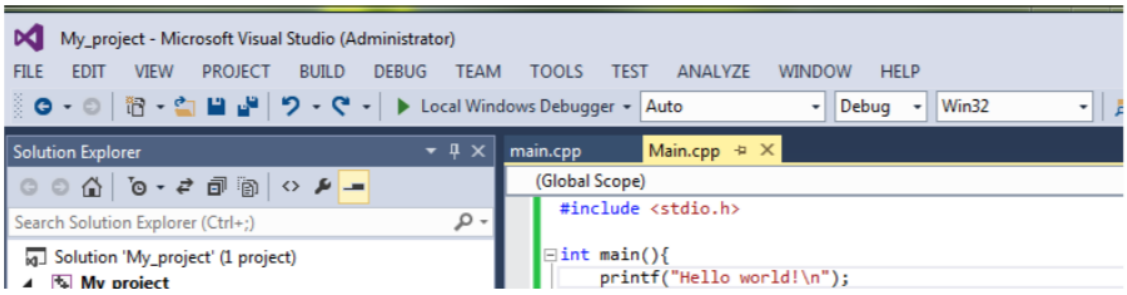

Fig 7: Running Code – from the toolbar

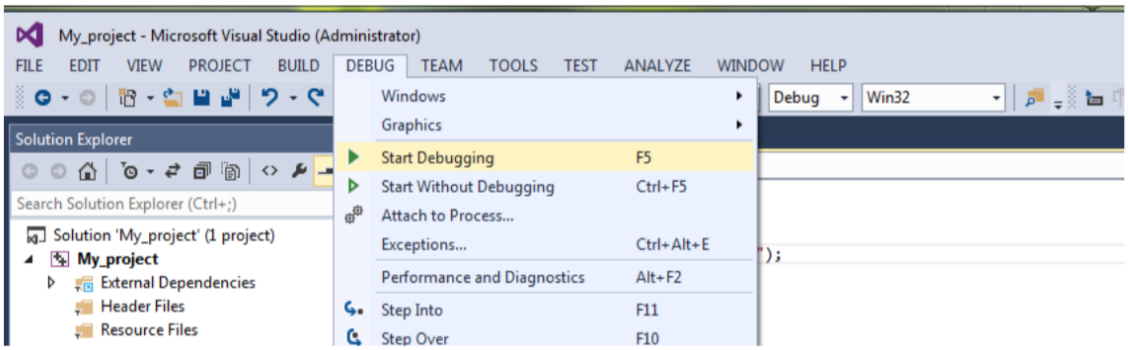

Fig 8: Running Code – from the menu

If you are quick, you should see a window pop up, display 'Hello World', and disappear again. This turns out to be a problem with the IDE – it's designed for professionals creating long pieces of code that are turned into Windows applications, and it can't really handle short programs that end in a few milliseconds. There are two ways around this problem, and the easiest one for now (because our code is *very* short) is to add a single function, the ***getchar*** function, which basically hangs around until the user types *return*. Change your code to the following, and try building and running it again.

```
#include <stdio.h>

int main(){
    printf("Hello world!\n");
    getchar();
}
```

If that works, you can use this technique at the end of any short piece of code, and you are ready to try all the real programming exercises in this, and future laboratories!

As an additional, very useful piece of knowledge, the other way to stop your program from finishing too quickly to see, is to add ***Breakpoints***. These are flags in the code to tell it to stop and wait. In addition, at a breakpoint, you can get the IDE to display the values of all the variables in the code. This turns out to be spectacularly useful in the debugging process. Add to your code so that it looks like the following (a bit of cutting and pasting should make this easy).

```
#include <stdio.h>

int main(){
    printf("Hello world1\n");
    printf("Hello world2\n");
    printf("Hello world3\n");
    printf("Hello world4\n");
}
```

Now just click in the left hand margin of the program to get the following red dot

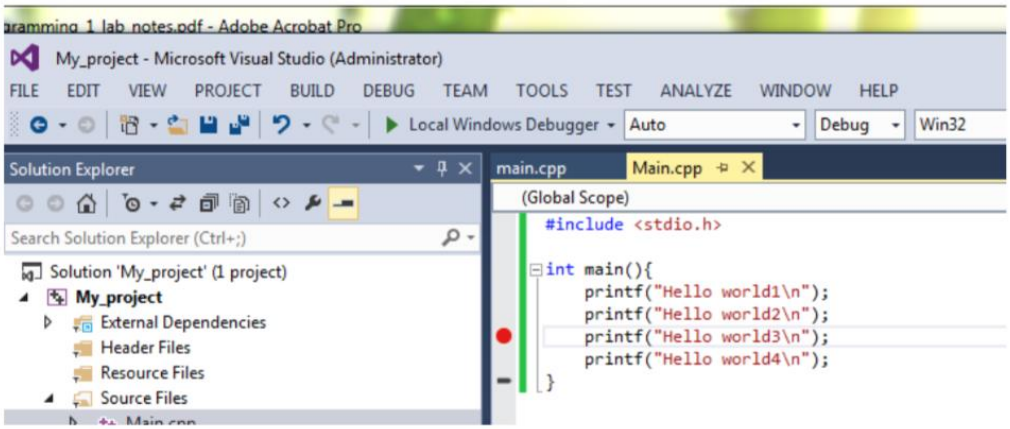

Fig 9: Adding a breakpoint

When you run the code, a new window will open (check to make sure that it hasn't opened in the background) like the following:

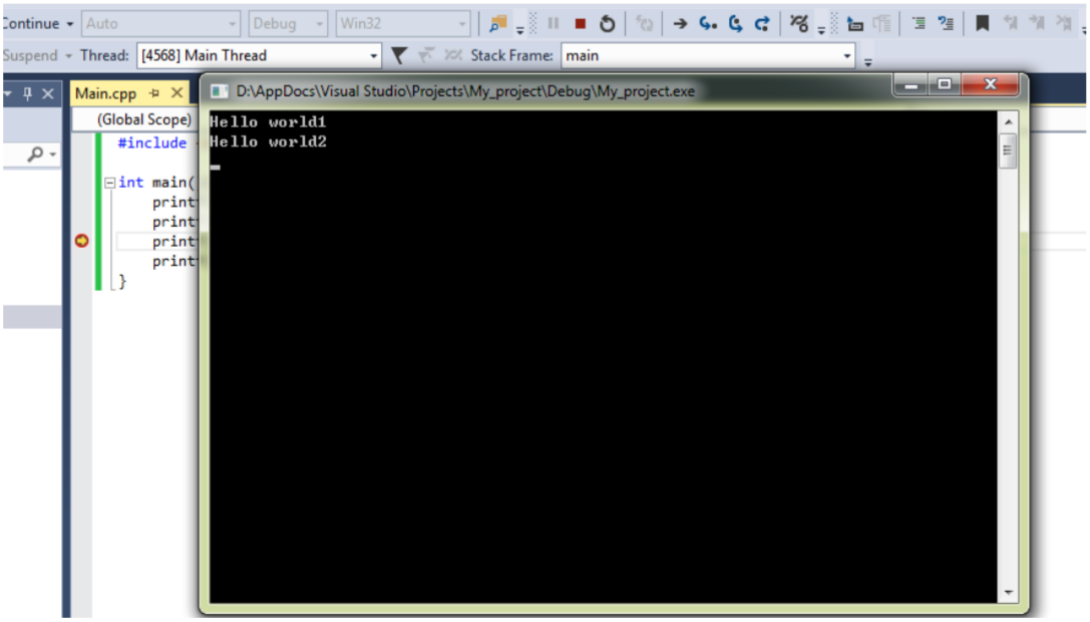

Fig 10: Result of a breakpoint

The code has halted at the breakpoint and is waiting for you to hit run (either through the menu, or the toolbar, or by F5) at which point it will resume from where it left off, and complete. Try right clicking on lines of code to experiment with setting, deleting, and using breakpoints.

Exercise 1: “Hello, world” and friends

Background

By convention, the first thing to learn in a new programming language is to print the text "Hello, world!". Although text isn't very exciting by itself, the ability to output text is vital for debugging (fixing programs).

Most programming languages, including C, indicate text with "double quotation marks": the first " indicates the beginning of text. The computer then treats everything until the next " as text.

This causes problems if you want to print an actual " symbol in the output. For that reason, we use *escape sequences* to represent symbols which are difficult (or impossible) to indicate with plain text.

In addition to telling a compiler how to create an executable file, source code should also tell future programmers why the code does what it does. If anything in the code is unclear, a programmer should add *comments* which explain the situation. More information about comments is on the good programming style page.

Technical details

Printing text (complete example):

```
#include <stdio.h>

int main() {
     printf("Hello, world!");

     // wait for a keypress
     getchar();
}
```

Selected escape sequences:

Sequence	Name
\n	New line
\t	Horizontal tab
\\	Back slash
\'	Single quotation mark
\"	Double quotation mark

There are two ways of adding comments (code exerpt):

```
// this is a single-line comment

/* this type of comment can span multiple    lines; everything is a comment
until    the    computer sees the ending */
```

Your task...

Write a short self-introduction (not necessarily truthful!), and a list of items you would buy if given 1, 10, 100, or 1000 pounds. Example output:

```
Greetings!  I am \Sir Percival\ of the Round Table, hailing from Vancouver,
Canada.  My fair city lies approximately 4,500 miles to the West. In my country,
we say ‘eh?’ a lot.

Free money, you say?  Well, let's see...
         Pounds   Purchase
         1        A packet of tea biscuits.
         10       DVDs of "Still Game", so I can learn Glasgow patter.
         100      An external hard drive for my laptop.
         1000     A visit to my family next Christmas.
```

Your program should:

- Print a message similar to the above example.
- No line should be longer than 80 characters -- you will need to use multiple printf() statements.
- Use all these escape sequences: \n \t \\ \' \"
- Use both types of comments: // /*...*/

(Optional: make it funny. We'll like you more. You won't get any marks for being funny, but hey, you'll make a friend! Looking at first-year programming exercises gets really boring.)

Once you have completed your task:

Show your work to a demonstrator/GTA.