CFJ PART 1 – BASIC C/C++

Exercise Sheet 1 – Introducing the g++ Compiler, and some very simple coding

There are many C++ compilers available, but throughout this module, we shall be using the C++ compiler developed by the GNU organisation called g++[1], which is installed on the department's Unix network.

Unlike Visual C++ and C++ Builder which can be considered Integrated Design Environments (IDE's) by providing project management facilities, pre-programmed code templates for rapid application development, integral debugging tools and class hierarchy browsers, g++ only compiles source code into executable files.

Unlike an IDE, g++ has only a command-line interface and does not have a sophisticated GUI. We will use g++ in this module because it is widely available (at no cost) for Unix/Solaris, all versions of Microsoft Windows, the Apple Macintosh and in a form that can generate executables for Palm Pilots. It also conforms to the C++ standard better than some more expensive compilers.

Visual C++ is available in the departmental labs which you may use if you wish for coursework, but all code must be compiled and tested using g++ before submission as all marking will be done with regard to the behaviour of code when it is compiled using g++ then run. We will not be doing any work in C++ that uses facilities that are not part of standard libraries, but using the rapid application development facilities of an IDE may result in code that will not run on the department's Solaris machines. Take Care!

To compile a simple program using g++, at the Unix prompt type the command:

`g++ filename`

Any professional will learn how to fully use their tools and will learn the limits and capabilities of the tools of their trade, be it an oscilloscope, football or whatever. The compiler or IDE you choose to use is the main tool of the software engineer's trade. You should spend time exploring how to use it. The on-line `man` pages for g++ will give a full user manual.

[1] In fact g++ is the name of script that calls the GNU compiler gcc. The gcc compiler can compile many of the C family of languages including C, C++ and Objective-C. By giving the command `g++`, we tell gcc which language we are using.

1. Using your favourite text editor (you do have a favourite don't you?), create a source code file containing the following code for the "Hello World"-style program. Save it as a file called **goodbye.cpp**. Compile it using g++ and confirm that a file called **a.out** has been created in the same directory as the source file. If the compiler produces any error messages, correct the source file, save it, then compile the file again.

```
#include <iostream.h>

int main () {

    cout << "Goooodbye Cruel world..." << endl;
    return 0;

}
```

2. You can specify the name given to the executable file by using the modifier `-o` when running the compiler. Recompile the helloworld program using the command

```
g++ -o goodbye goodbye.cpp
```

Confirm that the file **goodbye** has been created in the same directory as the source code file.

3. You have now seen and used the stream **cout** which names the standard output (the screen) and the "send to" operator **<<** which sends text and values to the stream object in order for them to be printed.

There is also a stream object called **cin** which names the standard input (the keyboard). There is an operator **>>** ("read from") which takes a value from a stream object and places it into a variable, e.g. **cin >>** `var;`.

Rewrite the program in exercise 1 so that it takes in a user-supplied message and prints this to screen (remember to use a char array).

4. Write a program that (a) reads in an arbitrary integer, (b) writes out its square root – make sure you check for negative.

Ingredients:

```
cout << "Type a positive intiger and press enter: ";
cin  >> value;
if (value < 0) ...
```

5. Now you're up and running try some looping code. Use a for loop to write a program that displays:

"*Why do I always do the same exercises for every language I learn?*"

ten times in the output window. Then modify this program to use a while and repeat loops with a counter.

6. Write a program that will use loops to print out the following triangle of numbers:

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
```

7. Once you have done this modify your program so that it is neatly formatted as shown here:

```
     1
    2 2
   3 3 3
  4 4 4 4
 5 5 5 5 5
6 6 6 6 6 6
```

8. The Fibonacci series is a number sequence, each of which is the sum of the two that precede it (starting with 1,1). Write a program to display the first 20 numbers in this series, with five on each line.

Fibonacci Series
1 1 2 3 5
8 13 32 53 85 . . . . .

9. Write a program that displays all of the factorials from 1 to 12 and displays them in a nice neat table. (N.B. Factorial of 4 = 4*3*2*1 for example)

Now re-write the program so it is as efficient as *you can possibly make it* (as few lines, variables and so on as possible).