Laboratory 1: Introduction to C++

This is a typical script that you will be working with at each laboratory session. To work with such a script efficiently, follow the guidelines below:

1. The script is not a step-by-step tutorial. It introduces the problem but it is your task to solve it.
2. If you get stuck, make sure that you read the *Help* section at the end of the document.
3. If you still have problems, ask the lecturer, demonstrator or fellow students for help.
4. Try to do as much work as possible in the class, where it is easier to get help.
5. If you need help when working at home, use the Blackboard discussion board. I tend to reply to my emails, but sometimes they get buried, so don't worry if I don't reply to your email and send it again. My office is Room 325, and Zainab's is ???
6. Finish all assignments at least a week before the deadline. If you get stuck, you will still have one week to ask for help.

Task 0: Using CMake

In the lecture, you saw yesterday that it can be extremely complicated to use the IDE to maintain the project files, etc. particularly if most students use Windows, some use Mac, and others use Linux. There is a ZIP file on Blackboard with 4 (almost) empty C++ source files. There is also a 'CMakeLists.txt' file. It is a good practice to organise your source code using directories, etc. Do not compile code in the same directory as your source code. It can get messy... Extract the ZIP file, and set up the compilation environment using `cmake`. In the GUI, the source directory corresponds to the direction in which 'CMakeLists.txt' is. The binary directory is where the code will be compiled. Often, we call this directory 'bin'. Once the paths are set, press 'Configure'. The first time you run the configuration tool, you have to select a generator. If you want to use MSVC++ in the lab, make sure to use *Visual Studio 14 2015 Win64*. For Mac OS X, you may want to use the Xcode generator. For Linux, you may choose Makefile. Once the configuration step is over, press 'Generate'. The project files are now ready in the 'bin' directory. You can compile the code using your preferred IDE.

Task 1: Adding integers

Your task is to program a command-line application that:

1. Reads from standard input any number of lines, each containing 2 integers separated by a space.
2. Adds the two integers together.
3. Prints the result to the standard output using the notation: $x + y = sum$.

4. Stops reading after encountering the end of file (to type the end of file symbol, press *Ctr* + *D*).

The application should use the input and output streams.

The C++ objects and methods that you need to use are:

```
ostream& operator<< (val)
istream operator>> (&val)
std::cin
std::cout
```

Look them up in the C++ reference to understand how they work.

Remember to include `iostream` header.

Also, remember to document your code (see the commented block at the start of each .cpp file).

Task 2: Adding floating point numbers passed as arguments

Your task is to program a command-line application that works exactly like the previous one, except:

1. Any even number of values are passed as command-line arguments rather than being read from the standard input.
2. The values passed are strings containing the floating point numbers instead of integers.
3. The program ends with an error if the number or type of command-line arguments is incorrect.

The C++ objects and methods that you need to use are:

- `cerr` for the standard output of errors (that's where error messages should be printed); and
- `double strtod (const char* str, char** endptr)` to convert a C-string into float. To learn about a standard function, it is always useful to go on `http://www.cpluplus.com/`. In the menu on the left-hand side, there is *Reference*. That's where you have to go. Hint, `strtod` is defined in the `<cstdlib>` header file. As an alternative, you could use the search button to directly go to the page about `strtod`. The advantage of the search method is that you don't need to know in which header the function is.

Task 3: Printing out strings

Your task is to program a command-line application that:

1. Takes any number of words as command-line arguments.

2. Prints out the number of arguments passed to the standard output.
3. For each argument passed, counts the number of vowels and dynamically allocates a new char array that can hold exactly N characters, where N is the length of the original array (including the 0 at the end) plus the number of vowels times two. To achieve this, you have to write a function called *isVowel* that will return 1 if the character passed as a parameter is a vowel, otherwise it returns 0.
4. Copies the original word into the new array, putting a dash after each vowel (eg. vision would become vi-si-o-n).
5. Prints out the resulting string in a new line.
6. Deletes the dynamic array.

The C++ objects and methods that you need to use are:

```
operator new[]
operator delete[]
```

Task 4: Bubble sort

Your task is to program a command-line application that:

1. Reads a single integer N from the standard input, where N is the number of floating point values to be sorted.
2. Reads exactly N floating point values from the standard input. The numbers are separated by a new-line character.
3. Sorts those values in increasing order using your own implementation of the bubble sort algorithm, swapping the values using the function *swap* written by you.
4. Prints out to the standard output the number of elements and then, in separate lines, all the numbers in the sorted order.

Example: Given the input file:

```
3
1.5
-2
0
```

your program should generate:

```
3
-2
0
1.5
```

Good luck!

1 Help

1.1 C++ resources

There are plenty of tutorials and resources for Java programmers who wish to learn C and C++. You are strongly recommended to research this topic on your own to gain better understanding of the differences between C++ and Java. This module is not meant to teach C++ but expects that you gain basic C++ skills as you solve the tasks at the laboratory classes.

Some on-line resources for Java programmers moving to C++:

1. Moving from Java to C++ — `http://www.horstmann.com/ccj2/ccjapp3.html`
2. C and C++ for Java programmers — `http://goo.gl/SxHYS`
3. C++ tutorials and documentation — `http://www.cplusplus.com/`

1.2 Installing CMake on your computer

It is strongly recommended that you install CMake on your PC so that you can complete your labs and assignments at home. You can find it at `https://cmake.org/download/`. Go to *Binary distributions* and choose your platform.

1.3 Installing an IDE on your computer

For the same reasons, it is strongly recommended that you install Visual Studio on your PC, Xcode on Mac, or Eclipse CDT on linux. Note that the installation of the development tools is often a difficult and time-consuming process. Do not assume that you can install the software a few days before deadlines. Install the software as soon as possible and let us know if there are any problems.

1.3.1 Windows: Installing Visual Studio Community

Download and install *MS Visual Studio Community* from

`https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx`

1.3.2 Mac OSX: Installing Xcode

This setup will require installing Xcode, which can take up to 2-3GB of your hard drive. Make sure that you have sufficient hard drive space before your proceed.

Install *Xcode* from App Store. *Xcode* is a free application for Mountain Lion and later versions of OSX.

1.3.3 Linux: Eclipse CDT

Download and Install *Eclipse CDT* from `https://eclipse.org/cdt/downloads.php`. You will need a Java virtual machine as it uses Java in the background.