Lab Exercises: LAB 3

(File I/O, STL, Operator Overloading)

General guidance:

1. You will need to log on to your Linux environment for this lab.
2. Use a text editor of your preference to edit the code. For today's lab you will not need anything more complicated.
3. For quick on-line reference use (a book is preferable): *http://www.cppreference.com/*
4. Use the g++ compiler at the command line: For example `g++ ex1.cc -Wall -o ex1` will compile the source file ex1.cc and will generate the executable code in the program file ex1.
5. You can run a program from the directory where you compiled it by typing (for the above example): ./ex1
6. Use the `mkdir` command to create a directory structure like, e.g. `your_home/CIP/labs/lab3`. If unsure ask for help. Then `cd` to `lab3`.

Exercise 1.

File I/O

Make sure you look at the additional material on File I/O in C++. The material includes a set of slides and sample code. Look at both.

Compile and run the sample code, make sure you understand what the code does. Modify the code to alter its behaviour.

Note that some of the sample programs require text files to already exist in your current directory.

Once you are comfortable with the code, proceed to Exercise 2.

Exercise 2.

Simple File I/O: Reading from files

There are basically three ways you could read the contents of a file: character by character (using `get()`, word by word (using `>>`) and line by line (using `getline()`). The point of this exercise is to use all three of them and to appreciate their differences.

- Open the file “`file1.txt`” for input only (the file should exist in your current directory, and have more than 1 lines of text in it)
- Add appropriate code that checks for any I/O errors.
- Read and display in `cin` the contents of the file using all three methods mentioned above.
- What differences do you notice between the three methods?

Exercise 3.

Write a program `change.cpp` that allows you to replace a specific character in a text file with a new character.

Specify a filename as a command-line argument, followed by the position of the character in the file that you want to change, followed by the new character you want to write.

Notice that the file should be opened for input AND output operations.

Hint: This exercise makes use of this form of the main function: `int main(int argc, char *argv[]).` You will also need to make use of the function `atoi()` – converts an alphanumeric to an integer.

To use the program after compilation, you should type `change testfile 6 x`, and this will change the 6th character of a file called `testfile` to ‘`x`’.

Include appropriate code in the program to check that only the correct number of arguments are given at the command line.

Exercise 4.

STL: Lists

Lists, unlike vectors, can only be accessed sequentially (i.e. no random access allowed). This exercise brings some of the elements of the STL together, and applies them to lists.

- Create a simple `list` of 10 integers. Each value in the list will be a randomly generated number
- Output the values of the list using an appropriate iterator to access the members of the list
- Now sort the list, using the `sort()` algorithm provided by the STL
- Print the sorted members of the list as previously
- Now move the iterator to the end of the list, and print the members of the list backwards, i.e. starting from the last member and proceeding to the first member. **Hint:** where does the pointer go to when you use the `end()` function in STL? This will determine the correct solution here.

Exercise 5.

Operator overloading

Look at the additional material on operator overloading, and especially at the sample code given at the course web pages. Make sure you understand what the code does. Compile and run the sample code that is provided.

Extend the code by defining the "==" operator for `vector2d`. (Think: what will the return type of this operator be?)

Add appropriate code in the main function to check your code.