EE273 - Engineering Design for Software Development 2

Week 3, Semester 2

Learning Outcomes:

- Ability to use containers in an OOP environment.
- Practical experience of using C++ containers : list and vector.
- Experience of Exception catching methods within C++

Preface

- The lab sheets are released one week in advance of the appropriate laboratory session.
- It is critically important that students prepare in advance of the actual laboratory sessions; reading through the lab sheets, mapping out draft solutions and indeed trying out sample code.
- Students are expected at arrive at each lab session fully prepared for the lab exercises; the lab exercise are not designed to be completed solely within the lab sessions but are designed to be completed by the end of each lab session if appropriate preparation takes place.
- If, for any reason, the lab is not finished within the Tuesday lab session, students should complete the labs as part of their own study; the labs are available outwith allocated EE273/985 slots for either advance preparation or for lab catch-up. There is a drop-in session on Friday afternoons at 15.00 – attend if you have pre-weekend questions to ask.
- This lab is a relatively brief lab that builds upon specific material from previous labs (7, 9 and 10) and students are encouraged to ensure that the appropriate exercises have been completed and documented.
- In laboratory 7 last semester, you were asked to define a structure, create instances of that structure, enter data into each instance based on what is read from a file, and put the instances in a list. In this lab, we will ask you to do something similar, but this time making use of objects and containers. You should already have had plenty of practice at defining classes and creating instances of them, i.e. objects. Material on containers was given in lecture 14.
- Do not attempt simply to modify what you did in lab 7 – you should create the code for this lab from scratch as a new 'project', though you can feel free to go back to lab 7 for hints or to copy relevant lines of your own code. (If you've been doing your log book properly, that should be the best place to look!)

Part 1: A Class to Store Marks

1. Define a class that includes a `string` to store a student's name, an array of 5 `floats` to represent the marks for individual subjects, a single `int` representing the number of subjects for which marks have been set and a `float` that will represent the average mark.
2. Decide whether these data members of the class should be `public`, `private` or `protected`. (On what are you basing your choice?). Include relevant constructor, destructor, setter and getter functions in the class, including one to work out and set the average mark.

Part 2: Reading From a File

3. Create a text file that includes, on each line of the file, the name of a person and a series of marks out of 100. Include at least 5 people in the file and up to 5 marks for each one. How will the program that finally reads the file be able to recognise each separate item of data on each line in the file? (Separated by spaces, tabs or commas?

Or by some other means? What if there are fewer than 5 marks on any one line of the file?). (This will be good practice for things you are likely to need to do in your project!)

Part 3: Creating a List using Containers

Two main types of container are provided within the standard C++ library – a `vector` and a `list`. Both have standard facilities available for inserting, deleting and 'iterating' through the set of objects.

As far as the code you write is concerned, they both seem quite the same (see lecture 14 and just use `#include <vector>` or `#include <list>` as appropriate). The main advantage of a list is that wholesale copying of the set of data doesn't need to take place each time an extra item is added or an item deleted.

4. Write a program to read in each line in the above text file and create an object for each 'student' in which you should set the name and the marks read from the file. Add each new object to a `list`.

Part 4: Manipulating a List

5. Add code to your program to 'iterate' through the `list` and write the names and average marks out to the screen.

Part 5: Checking Data

6. Go back to the code for setting the individual marks. Add further code to check that a mark is between 0 and 100 (inclusive) and writes an error message if it isn't. (What mark should be set instead? Should the user be asked to enter one from the keyboard?) (You could use `try`, `throw` and `catch` to do this – see lecture 14).

This represents the last of the formal lab exercises prior to the next test and the formal project. Students are required to go back over all previous 11 laboratories and review the material contained within. Also, the logbook should be fully annotated and updated in readiness for the test and the project work.