EE273 - Engineering Design for Software Development 2

Week 01, Semester 2

Learning Outcomes:

- Enhanced implementation of classes and objects in C++
- Further experience in specifying private and public attributes and methods
- Ability to use OOP features such as inheritance, function overriding and overloading, and association.
- Ability to perform comprehensive testing of input data methods.

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.
- Students are expected to have their logbooks (and printed lecture notes) with them at all lab sessions and to be actively taking notes IN THE LOGBOOK during the lab.
- When defining classes, it is important to consider what are appropriate types for data members, i.e. `int, double, string`, etc. Data member types can also be your own defined 'structures' or indeed other classes.
- Furthermore one should also decide whether a member function is sufficiently simple to be defined 'inline', or if an implementation should be defined in a source code module (a `.cpp` file) separate from the header (a `.h` or `.hpp` file) in which the class is defined.
- One good practice to get into when writing functions that perform operations or use data coming from somewhere else, e.g. input by the user, is to test whether the operation has been successful or whether the given data is valid. If the test then fails, a specific return value could be used indicate failure, and other code that called that function could take some appropriate action.
- Progressing through these exercises, students should document (incrementally) the classes and their relative relationships and member functions as they evolve using UML-like diagrams.

Part 1: Inheritance

1. In a new header file, define a class called `Person` containing three private, data members – `Name`, `Gender` and `Age`. Define setter and getter member functions (i.e. 'methods') for these members. As well as the required constructor(s) also include an appropriate destructor.
2. In the same header file, define a `Student` class, derived from `Person`, containing further three private data members: `StudentNumber`, `NumberOfSubjects` and a pointer to a string called `Subject`. Do not forget a `Student` constructor and destructor.
3. Define methods for `Student` to get and set the `StudentNumber`.
4. Define a further methods within `Student` that:
    i. prompts the user to input a number of subjects and assign to `NumberOfSubjects`;

ii. uses new to create an array of `strings` to which the `Subject` data member will point (the size of the array should be equal to `NumberOfSubjects`);

[Include appropriate code in the destructor to ensure that the string array is deleted when objects are destroyed.]

iii. goes round a loop asking the user for the values of each `string` in the `Subject` array.

5. Add suitable tests to your setter functions to test the validity of a value being set, e.g. to check that the student number is in the right range. Return a value from the function that would indicate a failure in validity testing.
6. Define second additional class derived from `Person`. The class is called `Staff` and contains two protected data members: `NumberOfSubjects` and a pointer to a `string` called `Subject` to represent the modules the member of staff teaches.
7. Implement code an appropriate method within `Staff` that:
    a. prompts for the `NumberOfSubjects`,
    b. creates a `string` array of size (`NumberOfSubjects`) to which `Subject` will point,
    c. add appropriate code to the destructor to ensure that the string array is deleted when objects are destroyed.
    d. and asks the user for the name of each subject.

(Consider how the classes (`Student, Person`) and methods developed previously could be used to derive the `Staff` class and methods).

8. In the main body of your code, i.e. `main(),` declare arrays of 4 students and 2 staff. Use loops and suitable calls to methods to enable the user to input the values of the variables relevant to each class (including the name, gender and age of the person).
9. Use further loops to display on the screen the names, student number and the subjects associated with each `Student`. Also, display on screen, for all `Staff`, the names and subjects taught.
10. Add code to prompt the user to be able to delete/remove a person (staff or student) from your list.

Demonstrate and explain to one of your group members, the operation of your code. The details of your group should be the exemplar data to run the code. Ensure that you have a UML representation of your solutions in your logbook that can then be adapted in the future exercises.

Part 2: Method Overriding

11. Add a new data member (`Mature`) to the `Student` class that indicates if they are a mature student or not. A mature student is deemed as someone who is over 26 year old. Add methods to the `Student` and `Staff` classes to set `Age` in the base class (`Person`). For the version of the function in `Staff`, simply write a message back to the console saying that it is impolite to ask a member of staff's age. In the version in `Student`, use the value of `Age` to set the value of `Mature`.

Part 3: Method Overloading

12. Add two pointers (to `int`) within `Student`, one called `CourseWork` and the other `Exam`. Add further code to the `setSubject()` method, created previously, to make each point to arrays containing `NumberOfSubjects` elements.

13. To Student, add two versions of a method called `setMarks()`. In each, use one argument to determine which element of the marks arrays should be assigned. In one version of `setMarks`, two further arguments are received and are used to set the values of particular elements in both the `CourseWork` and the `Exam` arrays. In the other, only one further argument is received and is singularly used to set a specific `Exam` mark.
14. Add code to `main()` to allow marks for each `Student` to be entered (via the keyboard) and call the correct version of the mark assignment method within `Student`.

Part 4: Association

This last section considers the use of association: the building of links between different classes. This is a more advanced concept and students are being asked, at the end of the exercise to reflect upon alternative (better) design choices for implementing a solution. Parts 1 to 3, represent an incremental progression to a solution. In this last section, you are required to consider a partial re-design so prior to writing any code, write down a revised UML representation of your proposed solution.

15. Create a `Subject` class having two data members: `Name` and `Code`. Define setters and getters for these.
16. Replace the `Subject string` arrays in `Student` and `Staff` by arrays of pointers to objects of type `Subject`. Create a setter function that takes a pointer to `Subject` as an argument.
17. Create a member function in `Subject` that returns a pointer to itself if a name given as an argument matches its own name.
18. Add code to `main()` to find a pointer to the correct `Subject` object when the name of a subject is entered for a student or for a member of staff, and set the pointer in the relevant `Student` or `Staff` object.

Consider the following:

Would it have been better to have defined another class called `Course` which points to a number of subjects, and to have a method which, for a given subject name, finds the correct pointer, rather than writing the code for finding a subject in main()?

Do you think having a `Course` class would enable efficient storage of other attributes?

Recording Progress

- Print out listings of your programs and put in your logbook. Make a note in your logbook of anything that you found difficult but learned how to do (so that you can look it up again later in case you forget it). Make a note of the answers to all key questions.
- Demonstrate the operation of your code with a colleague and test the program. How well does your program perform compared with theirs?

Write some general reflective comments about the laboratory and speak to the teaching staff to discuss your progress.