EE273 - Engineering Design for Software Development 2

Week 10, Semester 1

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

- This lab builds on previous lab in which you were asked to implement a structure to store a complex number and some functions to perform complex number arithmetic. In this lab, however, you will need to use material from the lectures that introduced object orientation in C++ and showed how to define and use classes.
- 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: Designing Objects

Before you start writing any code, think about what you might need for a complex number class – which data members and which member functions. What do you really need to store concerning the value of complex number? In what different forms might a user of a complex number want to see it? Do you need to store the relevant values in this alternative representation or can they be calculated – using the data that are stored and an appropriate member function – only when they are needed? What other calculations might the user of complex numbers want to perform? Examples might include finding the complex conjugate, returning the magnitude or phase of a complex number or adding,

subtracting, multiplying or dividing two complex numbers. Functions to do these could be added to your complex number class.

In your log book, write down, using UML, all the features of the class – what the data members will be, what the member functions will be and which of the data and functions will be 'public' and which 'private'.

Part 2: Declare and Implement Complex Number Class

In a `.hpp` header file that can be included in modules of source code, define a class for representation of complex numbers in 'standard' form (i.e. with real and imaginary parts). Include suitable prototypes for the member functions. Make the different data members and member functions 'public' or 'private' according to your design from part 1.

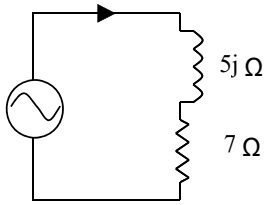
Include the header file you created in part 2 above in a `.cpp` module of source code. In that source code module, write the implementations of the member functions of the complex number class you defined.

Part 3: Using Complex Number Class

Create a function called from `main` that uses your complex number class to solve the simple circuit analysis problem below – use Ohm's law ($V=IZ$) to calculate the value of I.

Include code in this function to ask the user for the relevant parameters of the circuit to be analysed. (Both this function and main can be in the same module).

Vs = 100V $\angle 30^{o}$



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.