EE273 - Engineering Design for Software Development 2

Week 8, Semester 1

Learning Outcomes:

- Ability to define an appropriate project structure and good code layout
- Ability to split programs into multiple implementation and header files
- Ability to define custom data types and structures
- Ability to define functions
- Ability to write and call functions within a C/C++ program and passing parameters by value, pointer and reference
- Ability to pass and return structure pointers

Preface

- The lab sheets are released one week in advance of the appropriate laboratory session.
- It is key 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 exercises 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.
- Students are expected to maintain a logbook as part of the laboratory work – taking notes, recording code and reflective comments about their solutions and progress to-date.

Part 1: Define a Complex structure in header file

In a header file that can be included in modules of source code, define a structure for representation of complex numbers in 'standard' form (i.e. with real and imaginary parts that are each floating point or double precision).

Do not forget to include the header file *"boilerplate"* directives `#ifndef …, #define …, #endif`.

Part 2: Use header file

Include the header file you created in Part 1 and write a simple program that asks the user for the real and imaginary parts of a complex number, initialises one instance of the complex structure defined in the header file.

The program then prints complex number as: ***Real+j Imaginary***
For example, if the user enters 3 for the real and -5 for imaginary the program must print **3-j5**.

Part 3: Extend Complex Number Header File

Extend the header file you developed in Part 1 to declare the following functions (Note: the arguments of the functions must be pointers to one or two instances - as appropriate - of your complex number structure). The functions you write should do the following:

- return the magnitude of a complex number;
- return the argument of a complex number;
- return true or false depending on whether two complex numbers are equal or not.

Note: the function definitions shall be implemented in a separate `cpp` file with the same name as the header file. For example, if your header file is `Complex.hpp` then the function definitions should be in `Complex.cpp`.

Show your code working by allowing the user to input the values of complex numbers and see, on the screen, the results returned by your functions.

- Compile, Link and Run the program
- Test with appropriate sets of data – can you “break” your code?
- Give case studies for run-time errors – show data used, fault occurred and how it was fixed.

Part 4: Extend Complex Number module

Create additional functions in your module of source code developed in Part 3 above that do the following things below and allow the calling function to access the result:

- find the complex conjugate of a complex number;
- add two complex numbers;
- subtract one complex number from another;
- multiply two complex numbers together;
- divide one complex number by another.

Show your code working by allowing the user to input the values of complex numbers, choose an action to carry out and see the results of your functions on the screen. (How would your code perform if the user entered 0+j0?). Demonstrate your working code to a Teaching Assistant. Make suitable notes in your logbook of what you did.

- Compile, Link and Run the program

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.