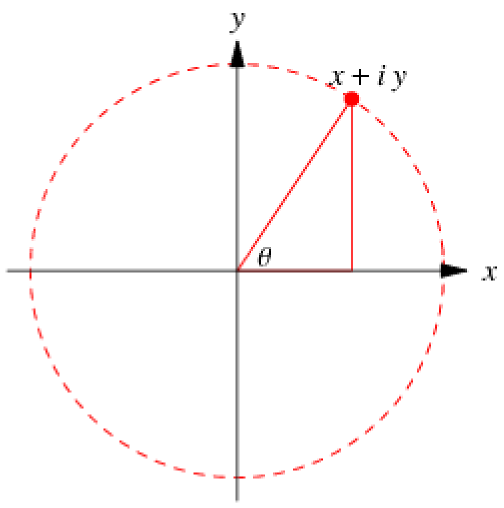
P3

Into the Imaginary Realm

In this lab, you will be constructing your own C++ implementation of imaginary numbers. Although calculating in imaginary number may seem intuitive to you, implementing imaginary number as objects in C++ requires some careful design and thinking. In other words, this lab forces you to think in an object-oriented way about a reasonably *complex* problem.



Schedule

Preparation time	:	3 hours
Lab time	:	3 hours

Items provided

Tools	:	None
Components	:	None
Equipment	:	None
Software	:	Eclipse and MinGW set up for C++ projects

Items to bring

Essentials. A full list is available on the Laboratory website at https://secure.ecs.soton.ac.uk/notes/ellabs/databook/essentials/

Before you come to the lab, it is essential that you read through this document and complete ***all*** of the preparation work in section 2. If possible, prepare for the lab with your usual lab partner. Only preparation which is recorded in your laboratory logbook will contribute towards your mark for this exercise. There is no objection to several students working together on preparation, as long as all understand the results of that work. Before starting your preparation, read through all sections of these notes so that you are fully aware of what you will have to do in the lab.

Academic Integrity – *If you undertake the preparation jointly with other students, it is important that you acknowledge this fact in your logbook. Similarly, you may want to use sources from the internet or books to help answer some of the questions. Again, record any sources in your logbook.*

Revision History

January 20 2020	Ivan Ling (il)	First revised version of this lab created

1 Aims, Learning Outcomes and Outline

This laboratory exercise aims to:

- Introduce the concept of using objects to represent abstract mathematical domains.
- Strengthen your grasp of object-oriented programming.
- Challenge yourself to build your own tools to solve mathematical problems.

Having successfully completed the lab, you will be able to:

- Make your own complex number library and use it to solve simple problems.
- Appreciate the concept of overloading and references.
- Use function overloading for arithmetic operation on your custom datatypes.

The overall objective of the lab is to create your own complex number class and use it as a custom datatype to solve problems involving complex numbers. Before starting this lab, you should brush up your understanding of complex numbers.

---

2 Preparation

Read through the course handbook statement on safety and safe working practices, and your copy of the standard operating procedure. Make sure that you understand how to work safely. Read through this document so you are aware of what you will be expected to do in the lab. Read up on operator overloading and how to overload build-in operators (+,-, *, /, ==, != , etc).

2.1 Background – Complex number operations

List down all the special operations which could be done on complex numbers. These are operations which could not be represented by the build in operators (e.g. Re(), Im() and Mod()).

2.2 Complex Number Class

(i) Design a class to represent the complex number with appropriate constructors. Your constructor should be overloaded appropriately with the correct overloading method. Implement your operators as member functions in your class. You may not need to think about the actual implementation of the function for now.

(ii) Design appropriate overloading schemes for out-of-class overloading support for basic build-in operators.

2.3 Impedance of circuits

For this lab, we will solve a simple problem involving the impedance of an RLC circuit.

(i) How do you find the real part of impedance in an RLC circuit?

(ii) How do you calculate the imaginary part of the impedance of an RLC circuit?

(iii) Write a simple function that takes in the total series resistance, capacitance, inductance and frequency of a series RLC circuit and convert it into impedance.

(iv) Why is impedance important in a circuit?

---

3 Laboratory Work

At the start of the session make sure that you can log on to the PC and can start eclipse and set the project to a C++ one. Code file extensions are .cpp and headers are .h. Have separate header file for declarations of functions from their definitions.

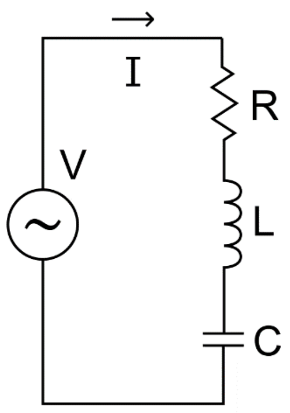
3.1 Implement complex class

1. Using the code prepared during your lab prep, write a simple program to demonstrate each of the functions in your class.

3.2 Convert RLC to impedance

1. Using the code from your prep, verify that your program can find the complex impedance of a series RLC circuit.

3.3 RLC circuit simulator



1. Create a new class to represent a series RLC circuit.
2. Your constructor should be able to take in all the necessary parameters to solve an RLC circuit.
3. Create a user interface for your circuit such that it prompts the user for the appropriate RLC values.
4. Prompt the user for Frequency and Amplitude of the voltage source.
5. Calculate the current of the circuit in complex form.
6. Calculate the phase difference between the voltage source and the current.

---

4 Optional Additional Work

Marks will only be awarded for this section if you have already completed all of Section 3 to an excellent standard and with excellent understanding.

Think of different ways to make a better representation of your circuit. For example:

You may create a separate class to represent Angle. This class should represent a number between 0-360^o, or 0 - 2π radians. After that, overload your complex class to allow the constructor to take in values in the phasor form (Polar form, $Z\angle\theta$). Subsequently, modify your class to allow conversion between polar and cartesian form ($Z\angle\theta$ to R + jX) and vice versa.

Appendices

References

http://physicstasks.eu/1540/series-rlc-circuit Sample question for Series RLC Circuits.

https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm Operator overloading

https://www.mathportal.org/formulas/pdf/complex-numbers-formulas.pdf Complex numbers