CS 2028C 001

Kyle Russell, russek5

9/18/2023

Lab 3 Report

Overview

In Lab 3, we explored the concept of abstract data types. More specifically, we created an implementation of the complex number system in C++ using our knowledge of classes. Within this class, we created operator overloads for addition, subtraction, multiplication and division, as well as the "==" comparison operator. The objective was to be able to do basic arithmetic with complex numbers as if they were integers or floats, and cleanly display their output in the same manner.

Task 1

The objective of task 1 was simply to create the class declaration as a header (.h) file. The purpose of this header file is to define the member variables, as well as the member functions. In this class, there are 2 constructors, 8 operator overloads, and two getter methods declared. The 2 constructors are a default, and an overload with both members assigned. Each operator overload takes a reference to a complex number or a double in the case of scalar multiplication and division, with the ostream operator taking a reference to an ostream object in addition to the complex number. The objective of passing by reference is to reduce the number of copy actions that the program makes and reduce both the compile time and the time that the functions take to run.

An issue I ran into was anything in the STL not being initialized, despite the file being included in the main file (Lab3.cpp). This is solved by adding the define guards and including the iostream library. Declaring the class is a simple, yet important part of the object oriented programming that C++ is most commonly used for, and it can help prevent things from being incorrectly defined, as well as improving compile time and reliability.

Task 2

The purpose of task 2 is to define the member functions of the class in a separate file (complex.cpp) in order to give them functionality in task 3. Using complex as a namespace, you can specify which function you're implementing from the class.

I.E. `double complex::getReal()`, and `complex complex::operator + (complex& number)`.

Task 3

Task 3 was to implement the actual function of the class, which turned out to be rather simple. First, create a resultant complex number. Then, using the reference of the number passed into the operator, do arithmetic on the real and complex components of the number respectively. This can be accomplished by using the "this->" pointer and passed number's member. Originally, I had implemented multiplication and division of 2 complex numbers, but the assignment was for scalar multiplication and division. I simply added the scalar operations to the header file and defined them in the class definition file and moved on.

There exist 2 constructors for this class. A default that initializes both members to zero, and one that initializes the members to the quantities you pass into the constructor. There are also two getters that serve to return the real or complex component. These ended up unused in the main program.

Task 4

The final task was to create a program to run the arithmetic on the complex numbers that I had just created. There are two functions created in the main file. One is an overload of the stream out operator, returning a formatted complex number to make the output easier, and the other is a function that returns a constructed complex number after user inputs. For the main logic of the program, I decided on a while true loop that will only terminate once you select the option to terminate the program. Within this loop, I define a double as a scalar, an int as an option selection, a bool for equality and 3 complex numbers.

When you input an option to select, it runs that choice into a switch statement that will determine what arithmetic to do on the complex numbers. Since this is a while loop, each case is terminated with a continue statement, wherein the loop is repeated until you choose the option to exit the program.

Final Thoughts

This lab is intended primarily to teach how operator overloads work, given that it's the bulk of the class definition. Being able to overload the STL operators is important for making operations easier on the programmer, and also making the program more efficient. Overloads are more elegant and effective than creating getter functions and operating on their returns. In each task, I made an edit to the class definition file as to fine tune its functionality, but generally, the declaration file was left largely untouched.