Project 03: Complex Numbers

Covered Concepts: classes, overloading operators, helper/non-member functions, math complex numbers, assert, writing tests

Project Description:

Complex numbers (sometimes referred to as imaginary numbers) are an important mathematical area of study and are widely used in science and engineering applications. A complex number, z, has the form: z = a+b*i*, where a and b are real numbers and $i=\sqrt{-1}$. The real_part(z) = a and imaginary_part(z) = b. Complex numbers are often graphically represented in a Cartesian coordinate system with complex number z = a + b*i* represented as a point (a,b) with the X-axis representing the real parts of complex numbers and the Y-axis representing the imaginary parts. As a starting point, to learn about the history, use and background of complex numbers the reader should access the Wikipedia website on complex numbers: http://en.wikipedia.org/wiki/Complex_number.

The C++ standard library includes a complex data type implementation and may be used by including <complex> in a client program. The Microsoft Software Developers Network (MSDN) starting webpage for the standard library of the complex class found in <complex> is: https://msdn.microsoft.com/en-us/library/0352zzhd(v=vs.120).aspx. In this specification we shall refer to <complex> as the specification, interface, and implementation of complex numbers for C++ standard library. Note that when the MSDN website refers to members, it means not only member functions (methods), but also free functions (helper functions) that include complex objects as parameters and/or return complex objects. Note also that <complex> specifies a template-implementation of complex numbers template <class T> complex (see MSDN website restriction on class T). The implementation of <complex> includes instantiations of T as allows class complex<double>, complex<float>, and complex<long int>. The class T represents the type used for the real and imaginary parts of a complex number. You are to implement complex numbers with ***only*** double real and imaginary parts without using the complex class in <complex>.

The essence of this programming project is to build your understanding of the design of classes without regards to a particular object-oriented language and to build your knowledge of C++ syntax and semantics, especially the use of C++ class semantics for building and representing classes. For this project you are required to implement a complex class that is a simpler version of <complex>. You must build a similar interface and implementation according to the specifications shown below. The class interface and bodies should be placed in files: "complex.h" and "complex.cpp."

Project settings should be "Console, x86, Debug settings".

Starting Code and Testing Requirements – P03ComplexNumbersDriver.cpp:

This file contains the start of the driver code. Only modification should be to add more tests.

You must write your additional tests in the indicated section of the main() function. Part of your grade will be based upon on the quality and extent of your test code. Use similar formatting to the tests already coded in "main". You must supply at least 10 additional tests and you must show for each test case the corresponding output. A good test suite always includes tests for side-effects.

"complex" Class Specifications :

You are to implement a representation of complex number objects as follows. Use the Cartesian ordered pairs representation using double as the data type of the real and imaginary parts. A class designer could also choose to represent complex number objects using the polar coordinate representation of (r, θ) with each also of type double. (The reader should consult Wikipedia site for more information about polar representation and related functions.) The class complex interface and data member implementation should be placed in "complex.h" created in your VS 2022 project. (NOTE: If you try to copy code from this pdf document into your VS 2022, there may be problems with character codes in the VS 2022 C++ compiler and editor. It is best that you copy into notepad and then copy from notepad to VS 2022.)

The declarations and definition should have the same general structure as:

```
class complex {

public: // interface for operators and member functions (aka, methods)
  …
private:
        double re, im; //Cartesian canonical form
        // private section may also include private helper functions.
}
```

Member Functions

The following member functions must also be provided with the following return types and signatures:

- *Constructors*
  - complex () // default constructor that should initialize to (0,0).
  - complex (double a) // constructor used for double-to-complex conversion
  - complex (double a, double b) // constructor of Cartesian coordinate representation
- *Copy constructor:*

  The default can be used since the members are double and the copy semantics for double are what we want for the implementation of complex numbers. However, here is what the signature should be. Even though the default copy constructor is provided and will work the way we intend, you should explicitly write it for this project.
  - complex(const complex& c) // copy constructor.
- *Assignment operation (=)*
  - complex& operator=(const complex & rhs)

    // copy values from rhs object and return reference to current object
- *Class complex member arithmetic and additional assignment operators +=, -=, *=, /=*
  - complex& operator+=(const complex& z)

    // like the assignment operator ("=") return reference to current object
  - Similarly, write complex& operator **op**(const complex& z) // where **op** is: -=, *=, /=
  - complex& operator+=(const double x) // for use of conversions

    // like the assignment operator ("=") return reference to current object
  - Similarly, write complex& operator **op**(const double z) // where **op** is: -=, *=, /=
- *Accessors:*
  - double real() const
  - double imag() const
  - double magnitude() const //also known as absolute value of a complex number

Helper and Non-member (Free) Functions:

The following helper and non-member (free) functions should also be part of the complex part of complex.h and/or complex.cpp, whatever is appropriate. Note that +, -, *, and / are **not** to be friend functions. Hint: Use +=, -=, *=, and /= to implement the respective binary complex arithmetic operators.

- *Binary Arithmetic Helper Functions:*
  - complex operator+(const complex& z1, const complex& z2);
  - complex operator+(const complex& z1, const double x);
  - complex operator+(const double x, const complex& z);
  - Similarly, do the same for operator ***op***'s three signatures and return types where ***op*** = -, *, and /.

- *Unary Arithmetic Helper Functions:*
  - complex operator+(const complex& z);
  - complex operator-(const complex& z);

- *Comparison Helper Functions:*
  - bool operator= = (const complex& z1, const complex& z2);
  - bool operator!=(const complex& z1, complex& z2);

  Note: the lack of ordering operators, e.g. operator<.

- *Input/Output Helper Functions:*
  - istream& operator>>(istream&, complex&); // input and parse complex literals of form: (a, b)
    // where a and b are both read as type double literals.

    Note: You must read past and ignore the comma, and parentheses: ',', '(' and ')'.

  - ostream& operator<<(ostream&, const complex&); // Output the complex number in the form: (a, b)

    Note: Do not forget to insert a comma character between values a and b and matching parentheses around

Other Helper Functions:

Note: These are free functions and not member functions. Also refer to meaning of these functions by referring to the above-mentioned Websites.

- double magnitude(const complex& z); // also known as absolute value
- double real(const complex& z);
- double imag(const complex& z);
- complex polar(const double r, const double theta); // Constructs a complex via polar coords.
- complex polar(const double r);
- complex conj(const complex& z); // Returns the conjugate of complex number
- double norm(const complex& z); // Returns squared magnitude (absolute) value of z
- double arg(const complex& z); // Returns arg (theta) value of complex number, z.

  Note: Must be calculated from internal representation. Undefined for z = 0. Use atan2( ) function from <cmath> to implement this arg() function.