

Program1

Due Oct 13, 2018 by 11:59pm **Points** 30 **Submitting** a text entry box or a file upload
Available Oct 1, 2018 at 3:30pm - Oct 14, 2018 at 12:10am 12 days

This assignment was locked Oct 14, 2018 at 12:10am.

Topic: Objects and Classes

Example: Please refer rational class for an example (Lab 1)

Goal

This programming assignment exercises how to construct abstract data types through implementing a **complex** class in C++. It also reviews operator overloading, and input/output including the friend concept.

Overview of complex Class

The **complex** class presents the complex number $X+Yi$, where X and Y are real numbers and i^2 is -1 . Typically, X is called a real part and Y is an imaginary part of the complex number. For instance, **complex(4.0, 3.0)** means $4.0+3.0i$. The **complex** class you will design should have the following features.

Constructor

Only one constructor with default value for Real = 0.0, Imaginary = 0.0; But it should construct the complex numbers with, 1) no argument, 2) 1 argument, 3) 2 arguments. Please refer the rat2.h file and rat2.cpp.

Data members

The complex has two members which should be real values. For example, $x+yi$, Real = x , Imaginary = y .

Member functions

Provide the following two member functions:

getReal

returns the real part of the complex.

getImaginary

returns the imaginary part of this complex number.

Math operators

The class must implement binary arithmetic operations such as addition, subtraction, multiplication, and division. You should be able to use operators (+, -, *, /).

Addition (+)

Add two objects. For example, $(a+bi) + (c+di) = (a+c) + (b+d)i$

Subtraction(-)

Perform complex minus operation

Multiplication(*)

Multiply the left-hand-side object by the right-hand-side and return **complex** object.

Division(/)

Divide the left-hand-side object by the right-hand-side object. You have to know the division of complex numbers. **Divide by zero error should be handled** in this method. That is, print an error message, and return the original one. Since it is an exception error, should not affect the following tasks. (Same for /= method)

Conjugate

The complex conjugate of the complex number $z = x + yi$ is defined to be $x - yi$. This function returns the conjugate of the input complex.

Comparison

The class must implement ==, !=.

Assignment

The class must implement +=, -=, *= and /=. ([Divide by zero error should be handled in /= method.](#))

Stream I/O

The class must implement the << and >> operators:

Input

Take two values as a real and imaginary value.

Output

The format will be: **$X+Yi$** , where **X** and **Y** are a **Real** and **Imaginary** value respectively. Of course, if either **X** or **Y** is **0**, it should not be displayed. However, if both **X** and **Y** are **0**, the output should be **0**. Also note that if **X** is 1, it should be printed out as **1**, and that if **Y** is 1, its output should be **i** . **Wrong examples:** $1+0i$, $0+2i$, $1i$, etc..**In the case of Y be negative, it should not have “+” between the two. For example, print $2-3i$, instead of $2+-3i$.**

Statement of Work

Design and implement a **complex** class according to the specification outlined below. The following is the **main()** function that will test the code. This driver file is in the Files--

>Programs/Program1/[complexDriver.cpp](#) .

```
#include <iostream>
```

```
#include "complex.h"
```

```
using namespace std;
```

```
int main( ) {
```

```
    complex c1, c2( 1.2, 4.9 ), c3( 2.2, 1.0 ), c4( -7.0, 9.6 ), c5(8.1, -4.3), c6(0.0, -7.1), c7(6.4), c8(0.0, 1.0),  
    c9(0.0, 4.1), c10(0.0, -1.0), c11;
```

```
    cout << "type two doubles for c11: ";
```

```
    cin >> c11;
```

```
    cout << "c1 = " << c1 << endl;
```

```
    cout << "c2 = " << c2 << endl;
```

```
    cout << "c3 = " << c3 << endl;
```

```
    cout << "c4 = " << c4 << endl;
```

```
    cout << "c5 = " << c5 << endl;
```

```
    cout << "c6 = " << c6 << endl;
```

```
    cout << "c7 = " << c7 << endl;
```

```
    cout << "c8 = " << c8 << endl;
```

```
    cout << "c9 = " << c9 << endl;
```

```
    cout << "c10 = " << c10 << endl;
```

```
    cout << "c11 = " << c11 << endl;
```

```

cout << "c1 + c2 + c3 = " << c1 + c2 + c3 << endl;
cout << "c7 - c8 - c9= " << c7 - c8 - c9 << endl;
cout << "c2 * 22 = " << c2 * 22 << endl;
cout << "c2 * c3 = " << c2 * c3 << endl;
cout << "c2 / c3 = " << c2 / c3 << endl;
cout << "c2 / c1 = " << c2 / c1 << endl;
cout << " c2 / c5 = " << c2 / c5 << endl;

cout << "c4 == c4 is " << ( ( c4 == c4 ) ? "true" : "false" ) << endl;
cout << "c4 != c4 is " << ( ( c4 != c4 ) ? "true" : "false" ) << endl;

cout<< "Conjugate of" << c5 << " is " << c5.conjugate()<<endl;
cout<< "Real of" << c3 << " is " << c3.getReal()<<endl;
cout<< "Imaginary of" << c4 << " is " << c4.getImaginary()<<endl;

cout << "(c5 += c2) += c3 is " << ( (c5 += c2) += c3 ) << endl;
cout << "(c5 -= c1) -= c2 is " << ( (c5 -= c1) -= c2 ) << endl;
cout << "(c5 *= 22) *= 13 is " << ( (c5 *= 22) *= 13 ) << endl;
cout << "(c5 *= c4) *= c4 is " << ( (c5 *= c4) *= c4 ) << endl;
cout << "(c3 /= 2) / c3 is " << ( (c3 /= 2) / c3 ) << endl;
cout << "c4 is " << c4 << endl;
cout << "(c4 /= c1) / c1 is " << ( (c4 /= c1) / c1 ) << endl;
cout << "c4 is " << c4 << endl;
}

```

Deliveries

Clearly state in your code comments any other assumptions you have made. Assumptions about **complex** members are placed in the class definition (.h file).

Turn in:


1) output (as text file, or captured image of console): This is the result with a given test file ([complexDriver.cpp](#) ) above.


2) complex.h file

3) complex.cpp

option) your own complexDriver.cpp only if your implementation is incomplete.

If your program is not compiled even with your own driver file, you will get zero point no matter what.

If you forgot to turn in your own driver file, and yours is not compiled with my driver file ([complexDriver.cpp](#) ) , then you get zero point.

If your program is compiled with my driver file ([complexDriver.cpp](#) ) then you do not need to submit your own driver file.

Make sure that your program can be compiled and executed on Linux.

Grading Guide

1. Correctness (26)

Compilation errors in Linux (0)

Successful compilation but less than 10 methods implemented(0)

Successful compilation with at least 10 methods implemented(10) + Correct output(16)

-1 per a wrong output or a wrong implementation of each method (including constructor).

2. Program Organization (4)

Proper comments (**Every method has to be commented** in the header and implementation file)

Good (2) Poor(1) No explanations(0)

Coding style (**proper indentations, blank lines, variable names, and non-redundant code**)

Good (2) Poor(1) No explanations(0)