Assignment Overview

This project focuses on some mathematical manipulation. It is worth 5 points (0.5% of your overall grade). It is due Monday, Sept 11th before midnight.

The Problem

You should all have learned something about complex numbers in your travels. A complex number actually consists of two parts: a *real* part and an *imaginary* part. The imaginary part is multiplied by the special value *i*, the √-1. A complex number is usually written in the form 1 + 2i , a sum with the real part first, the imaginary part second followed by i.

We are just going to do some basic math. You will read in two complex numbers and provide the sum, difference, product and quotient of those two values.

Some Background

You forgot about complex numbers, didn't you. Well, Wikipedia is here to help. There's far more information here than you need, but the two most important sections are https://en.wikipedia.org/wiki/Complex_number#Addition_and_subtraction and https://en.wikipedia.org/wiki/Complex_number#Multiplication_and_division . You can get the formulas from there

Program Specifications

Your program will do the following:

1. Take as input four floating point values (in this order, as indicated):
    a. a real value and an imaginary value for the first complex number
    b. a real value and an imaginary value for the second complex number
2. Print the following four results: the sum, the difference, the product and the quotient of the 2 complex numbers. You will use ***exactly*** the following format:
    a. Each result on a single line (thus 4 lines printed)
    b. The precision is to two decimal point of accuracy (see note 1c below)
    c. output looks like the following ***exactly***: `1 + 2i` that is, a float, a space, a plus sign, a space, a float followed directly by the letter i

Deliverables

proj01/proj01.cpp . The name of the directory is proj01, the name file you turn in should be exactly proj01.cpp. It will be checked upon handing it in to Mimir for the first test case. Just like the lab, you must click on "Project 01 – complex number manipulation", the ***parent*** of the directory proj01, to submit the file.

Assignment Notes:

1. We might as well try to make the output somewhat readable. You can look this up in the text or on the internet, but you can use the following modifiers that affect how numbers print.
    a. `cout << fixed`

Elements will be printed as floating point numbers (ex 123.456). Use this for the project.

b. `cout << scientific`
Elements will be printed in scientific notation (ex 1.23456 x 10^2). This is an alternative but ***not what we want*** for this project.

c. `cout << setprecision(2)` Floating point numbers will have 2 values after the decimal point and will be rounded, (123.46). To make this work we need another include, and that is
`#include<iomanip>`
Provide the `include` and use `setprecision(2)` for this project.

d. Thus `cout << fixed << setprecision(2) << 123.4567 << endl;` will print 123.46

2. The following statement will read two variables off of the same, space separated line. It is an example of chaining input:

```
…
double d1, d2;
cin >> d1 >> d2;
…
```

You can also do it on separate lines. Your choice

```
…
cin >> d1;
cin >> d2;
…
```

3. There are 4 tests provided. The first is a file-exists test to make sure you got the directory issue correct, the remaining three are input-output tests.
4. You ***do not*** have to check for bad input values. In general, we will explicitly indicate the errors we are looking for, but for now we are not checking for input errors.
5. If you read off of a single line of 4 space separated floating point numbers for the input, you can do a handy trick off the command line. You can redirect a file (containing that single line of input) to your main program. With that file in the same directory as the compiled `a.out`, you can do:

```
./a.out < fileOfInput.txt
```

This will automatically feed the input to the `cin` statement and produce the output. Makes it easier to test things, saves you from typing 4 numbers a lot.

6. You can check your calculations on http://www.wolframalpha.com/ or using python or some other approach that you are comfortable with. If wolframalpha, make sure you enclose the complex number in parens as in: (1 + 2i) + (1 + 2i)

Getting Started

1. In Mimir, I create the directory `proj01` and place in it an empty file (no contents) with the correct name, `proj01`. You can update the file contents there.
2. If you are in a CSE lab or on x2go (or whatever your environment), then bring up an editor and a terminal, create a project directory of any name (I would suggest the Desktop as that is easiest to work with) and create proj01.cpp

3. Write your code and compile it.
4. Prompt for some of the values and print them back out, just to check yourself.
5. Check that your calculations are right (as indicated above).
6. If you develop on x2go (or wherever), the easiest way at this point to check yourself on Mimir is to copy and pasted rom your editor to the empty file on Mimir and submit.
    a. You don't have to pass all the tests the first time! You can add more information and pass more of the tests as you progress. Do things incrementally.
7. Now you enter a cycle of edit-run to incrementally develop your program.
8. If a version you submit and test on Mimir passes all the tests, you are done. If time runs out and you didn't pass all the tests, then however many you passed indicates what grade you got for the project. Though it doesn't matter much now as this is relatively easy, at the end you do the best you can and pass as many tests as possible. However, ***you always know*** how you are doing and that is the advantage of Mimir.
9. ***Remember***: In the end, it only matters that it compiles and runs on Mimir. If it doesn't compile there, then it doesn't compile at all (no matter what else you did on whatever environment you used). Mimir is the last word.