CS165 Assignment 6

Jason Dorweiler

November 16, 2013

1 Requirements and Possible Solution

Part 2a.
This part asks to complete question 8.7 from the book. To do this I'll first have to create a prime number class that stores a prime number. By default this will be the prime = 1. There will also be another constructor that allows the user to set the the default values to another prime. I'm not sure if this default needs to be checked if it is actually a prime number but I think it might be a good idea to set it up to do that. After that I have to create two functions ++ and – that will increase and decrease the prime number to the next prime. Then I will create a small test program for the class.
I found a neat way to check to see if a number is prime on this stackexchange post:
http://stackoverflow.com/questions/5200879/printing-prime-numbers-from-1-through-100

```
for(int i = prime+1; i < INT_MAX; i++)
  for(int j = 2; j < INT_MAX; j++){
        if(i % j == 0){
            break;
        }
        else if(j+1 > sqrt(i)){
            this->prime = i;
            return i;
        }
  }
}
```

I made some small changes to check for only numbers larger than the number we currently have as the prime number. Then it's easy enough to just put everything into it's own class.

Part 2b.

For part two I have to write a class that works with rational numbers. The class will take two integers to represent the rational numbe. The class needs constructors that set the member variables. There needs to be another constructor that will take in a single integer and return a whole number. Another constructor that defaults to 0/1. Then another function to overload the << and >> operators to print out the numbers as a fraction with negatives. Then overload the following operators: $==, <, <=, >, >=$ $, +, -, *, and/$. And finally write a test program for all of it. I noticed that for the logical operators I only need to write functions to check ==, and <. All of the other logical operators can be determined by just using those two. For example: using ==, and < to make >=

```
if(equal || !(lessthan)){
  return true;
}
else{
  return false;
}
```

Part 2c.

For this part I will have to do the same as part a and b but this time building a complex number class. The complex number class should be a double.

Part 2d.

For the complex number class I will have to implement several things. First the class will have to have the following constructors:
Default constructor: This will just be a constructor that has the option to set starting values for the real and imaginary variables. If nothing is entered then it will just initialize the two variables to zero.

Two arg constructor and default second parameter: This constructor will take two double variables but with the option of the second one having a default value. For example:

```
complexClass(double real, double imaginary = 0}(
  this->real = real;
  this->imaginary = imaginary;
}
```

Then I will have to overload several operators: $+, -, -unary, *, /, >>, and <<$. The class also needs to use the following member functions: real, imag, abs, norm, conj, and polar. I'll probably need to look up how each of those member functions should work as I go. My complex math is a bit rusty.

Then finally write a test program and compare the output to the std::complex class.

2 Implementation

The implementation for the prime variable class was pretty straight forward. I overloaded both of the increment and decrement operators and added in a default constructor to set the prime to 2. The decrement operator checks to see if the prime value is 2 to keep from returning numbers below 2. I also included a isPrime() function to check to see if the number entered by the user is actually a prime. If the number entered is not a prime then this the prime number gets set to the next highest prime number.

For the rational number class I made two constructors. The first one is just a default one that sets the numerator to one and the denominator to 1. The second constructor takes in a single whole number and sets that to the numerator and sets the denominator to 1:

```
RationalNumber(int wholeNumber){
    this->numerator = static_cast<int>(wholeNumber = 0);
    this->denominator = 1;
}
```

After that I have a bunch of friend functions. The books seems to prefer using friend functions so I stuck with that way of doing things. I made input and output stream overloads, and all of the overloaded operators. I also made private normalize function. I think making it private is the right way to do this since there is no need to have access to this function out side of the class. It works by checking if the denominator is less than zero. If it first just flips the values of each. Then it reduces the fraction to is lowest form.

```
void RationalNumber::normalize(){
        // if the denom is negitive then flip the values
        // to make the numerator negative
         if(denominator < 0){
            denominator = -denominator;
            numerator = -numerator;
        }

        // new temp integers for the numerator and denominator
        int x, y;
         x = abs(numerator);
         y = abs(denominator);

        // This loop goes through the y=x%y calculation untill y = 0.
        // it then stores the x value to reduce the fraction to the lowest
```

```
        // form.
        while(y){
            int t = y;
            y = x%y;
            x = t;
        }

        // use the x calculated above to reduce the fraction.
        if(x > 1){
            numerator /= x;
            denominator /= x;
        }
    }
```

For the complex number class I made three constructors. One default that takes no arguments and sets the real and imaginary variables to zero. Another that sets the imaginary variable to zero by default and the real variable to whatever was entered. This allows someone to enter in just a whole number instead of a complex number. The third constructor I never completely got to use. This constructor sets the real and imaginary variables to be rational numbers. I though this would be nice to try to get the fractional numbers to print out as rational numbers using the rational class instead of just printing as floats.

```
ComplexNumber( RationalNumber& r1,  RationalNumber& r2){
    Rat1 = r1;
    Rat2 = r2;
}
```

I was having problems getting it to properly print out though. When I would use the overloaded output stream << it would still try to use the overload for the complex class instead of the rational class. I did a bunch of searching online and couldn't find anything to help but I left it in just in case I figured something out.
I also made all the required overloaded input, output, and operators. All of the member functions work as they should and I compared them to the std::complex class.

3 Testing and Debugging

I made a lot of use of the gdb debugger for this lab. I actually didn't run into too many actual bugs but it was fun to play around with the debugger. For the complex number class is was good to check my output against the output from the std::complex class to make sure that I was doing things correctly.

The prime number class asks for a prime integer to be entered. If a non-prime is entered it will check and increment it to the next highest prime number. It does the same if a float is entered. One limitation is that it only goes up the max size of an integer using $INTMAX$ from limits.h. If the user enters characters instead of a number it will automatically get set to 2.

The rational number class asks for two integers for the numerator and denominator. It will do some error checking and output a warning if the user enters anything other than an integer. The class will also use the normalize function to automatically reduce the input into the reduced form.

The complex variable class asks for two doubles for the real and imaginary variables. Since they are doubles it will accept both integers and floats. One problem I ran into with this class is the polar function. I checked my polar function and it works and gives me the same polar conversion that I get from Wolframalpha.com. I can't get the std::complex polar function to work correctly. The c++ website shows that it takes polar(magnitude, angle) which I've tried to do using:

```
polar(norm(stdComplex1), arg(stdComplex1))
```

Where norm should give the magnitude and arg should give the theta angle but it doesn't seem to give me the same values. Either way I know my class works and is a little easier too.

4 Reflection

I think the hardest part of this assignment was that pretty much none of it was covered in the actual week 6 materials. I got through it by reading the book and whatever I could find online. The week 7 videos seem to cover the exact stuff I needed for this assignment but I had already completed most of it by the time they were posted. It seems like the videos and assignments/labs are not in the right order.

One other thing that was a bit confusing to me is how to set up the actual class.h file. It seems like some books/people put just the function prototypes in the header file and then have a separate cpp file for the member functions. I've also seen others that include the member functions in the header file. I'm not sure if there is a more correct way of doing it so I just went with keeping the member functions inside the header file.