QuadPoly class and test program

50 points - Due Sunday, April 29, 11:59 PM

Assignment

For this assignment you will implement a simple QuadPoly class and a simple program that uses it. This assignment incorporates many of the concepts we've talked about this semester; nothing in here is new or complicated, but there are a lot of pieces! Make sure to read this project carefully so you get everything.

The QuadPoly class

The QuadPoly class is a representation of a quadratic polynomial. If you remember back to your algebra days of yore, that's a second-order polynomial - one that looks like this:

$$ax^2 + bx + c$$

Here's what the class declaration for the QuadPoly class should look like:

```
class QuadPoly
{
   public:
      QuadPoly();
      QuadPoly( const QuadPoly& qp );
      QuadPoly( int a, int b, int c );

      int countSolutions();
      float getFirstSolution();
      float getSecondSolution();

      bool operator==( const QuadPoly& qp );
      QuadPoly& operator=( const QuadPoly& qp );

      friend ostream& operator<<( ostream& os, const QuadPoly& qp );

   private:
      int a, b, c;
};
```

Constructors (5 pts)

Implement the default constructor (which should set a, b, and c to zero), the copy constructor, and the constructor that takes three integers (which should be assigned to a, b, and c). The copy constructor should be implemented by invoking operator=.

Operators (5 pts)

Implement operator== and operator=. These are extremely straightforward - check the notes for examples on how to do each of these.

Solving Functions (10 pts)

Implement the **countSolutions()**, **getFirstSolution()**, and **getSecondSolution()** functions. These methods should do pretty much what you'd expect: the **countSolutions()** method tells you how many (real) solutions there are for this quadratic equation, and **getFirstSolution()** and **getSecondSolution()** return them. These last two functions assume that a real solution does exist - if there's only one solution, then they both return the same value.

For this project, you can ignore complex solutions. This is some basic algebra... search for "quadratic equation" on Wikipedia or Google if you need a refresher. The value of the discriminant (b^2 - 4ac) tells you how many real solutions there are, and the usage of the quadratic equation itself shouldn't present any problems.

You'll need the **sqrt()** function - **#include <cmath>** to get access to it.

operator << function (5 pts)

Overload the operator<< function so that you can directly pass QuadPoly objects to ostream objects (like cout or ofstream). The output should look something like this (depending on the value of a, b, and c):

```
2x^2 + 4x - 30
```

In this example, the value of c is -30. Give your function a bit of intelligence when it comes to negative coefficients, so that the output looks like this and *not* like `2x^2 + 4x + -30`.

Main Program (25 pts)

Your main program should instantiate an STL **vector** of QuadPoly objects. The program should repeatedly ask the user whether they wish to add a new quadratic, or solve the existing quadratics and quit. If the user opts to add a new quadratic, get input integers **a, b,** and **c** from the user and construct a new QuadPoly. Then use the STL **find()** function to see if the QuadPoly is already in the vector - if not, add it. **Each QuadPoly in the vector should be unique.**

If the user chooses the other option, your program should iterate through the vector and use the solving functions you coded above to solve each QuadPoly. The output should show the equation (using the operator<< function you defined), the number of real solutions, and the solutions themselves.

When the user chooses to solve and quit, the output should look something like this:

```
There are 2 quadratic polynomials in the list.
2x^2 + 4x - 30:   two solutions (3 and -5).
1x^2 + 2x + 3:   no real solutions.
```

Assignment Notes

- Remember, write and test your code in pieces! Get one thing working before you move onto the next thing.
- *Test* the solving functions - feed it sample equations you know the answer to, and see if it gives you the right answers. In **getFirstSolution(), and getSecondSolution()** it might be a good idea to convert **a, b,** and **c** to float values *before* you do any calculations on them... remember that if you mix floats and ints, sometimes the compiler is implicitly typecast things and sometimes the ints get truncated.

What to turn in: a directory containing...

- Your code, of course!
- A README file containing any relevant info, such as your name and compilation instructions (include the exact command you used to compile your code). Again, please include in the README the approximate amount of time you spent on this assignment. Also, the better you document your program/design in the README, the easier it will be for me to see what you were doing. This is always a good thing. :-)

Non-specific Notes

- Start early; ask questions.
- Refer back to the syllabus for hints about writing good C++ programs (or programs in any language): only work on small pieces of code at a time, and test them well before moving on. Compile lots, and fix the warnings and errors. Write lots of comments.
- **Be sure your code compiles on the CS department's Linux machines before turning it in**. Even if you write it elsewhere (on a different OS, a different IDE, etc), give it a quick compile and test on a lab machine before you turn it in. Remember, if your code does not compile, it will get (at most) 50% credit.
- You must submit your code using the department's electronic submit procedure. A link to instructions for doing this is on the class website. Our course number (for the purposes of the submit program) is **c109**.
- Just a reminder: the CLAS academic (dis)honesty policies will be enforced. You're allowed to discuss the assignment with others, but not to collaborate or share code. Nor are you allowed to find or use code off the Internet.