Homework 6: Templates

CSCI 60

Due before class Wednesday, February 19, 2020

Krehbiel

Overview. This homework will have you define a template function and a template class. You will also design a `Rational` class similar to that considered in class (but with an additional computational problem solving task) to use to test your functions, and you should be using your main function to test as you develop.

Submission instructions. Upload $\boxed{\texttt{hw6main.cpp}}$ with all tasks fully implemented. This week all your classes and test code can go in the same file, and compilation will be tested with the command

```
g++ hw6main.cpp -std=c++11
```

Part 1. Write a template function that overrides the insertion operator << for a vector containing elements of any type `T` so that each entry in the vector is output in order, separated by commas. Your implementation can assume that << is overloaded for elements of type `T`. E.g., if I had a simple class `Name` that stored a first name and a last name with accessors `getFirst` and `getLast`, I could overload << for `Name` as follows:

```
ostream& operator <<(ostream& out, const Name& n) {
  out << n.getFirst() << " " << n.getLast();
  return out;
}
```

With << overloaded for `Name` as above, your templatized vector overload for << would allow me to call `cout << v;` for a vector declared as `vector<Name> v;` to cout a comma-separated list of names.

Part 2. Design a template class for a heterogeneously-typed pair of elements `x` and `y`. Since `x` and `y` can be different types, your class template prefix should declare two typenames. You should have a 0-argument constructor that constructs default values for `x` and `y`, a 2-argument constructor whose arguments specify values for `x` and `y`, const accessors `getX` and `getY`, and + overloaded to construct and return a new pair whose `x` is the sum of the `x` values of the two operands and whose `y` value is the sum of the `y` values of the two operands. Your implementation will assume that + is overloaded for the elements of both types. Finally, overload << for your pair class to output a pair's values comma-separated and surrounded with parentheses, e.g., `(xstr, 2)`. You can assume << is overloaded for both types.

Part 3. A `Rational` class is defined and mostly implemented for you in the starter code. Finish the implementation by defining the following functions:

1. Define `reduce` to store the rational in reduced form. This function should (possibly) update the numerator and denominator of the rational without changing its value so that the denominator is nonnegative and the numerator and denominator have no common factors.

2. Overload + as a member function to return a new reduced rational whose value is the sum of the operands. Recall that fraction addition works as follows:

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + cb}{bd}$$

3. Overload << as a non-member function to print out the rational as a fraction, e.g., `2/3`. If the denominator is 1, it should just print the numerator as a whole number, e.g., `-2` instead of `-2/1`.

Part 4. Write code that tests your above code in the following way:

- Test Part 1 by first declaring a vector `intVec` to hold ints, then inserting the values 2, 4, 4, -1, and finally calling `cout << intVec << endl;` to ensure the console shows `2, 4, 4, -1, `.
- Test Part 2 by constructing two pairs whose `x` values are ints and whose `y` values are doubles corresponding to the points (1, 0.6) and (4, 0.8). Add these points together and cout the new point to check that the console shows `(5, 1.4)`.
- Test Part 3 by constructing and then console-outputting a rational with numerator 18 and denominator 66 to make sure it outputs `3/11`; further test that 3/-6 outputs `-1/2` and -8/-4 outputs `2`.
- To tests the templatization of Parts 1 and 2 using Part 3, declare a vector to hold rationals, insert the rationals from the previous test, and check that you can cout the vector to display `3/11, -1/2, 2, `.
- Finally write your own test code to check that you can store and correctly output a vector of pairs in which x is a double and y is a rational.