CSE 3150 — C++ Essentials — Yufeng Wu — Fall 2024
Programming Assignment 6. Due: 11:59 pm @ 10/11/2024.

There are objectives in this assignments.

1. Get familiar with STL algorithms. This is an interesting aspect of C++. You will learn how to use STL algorithms.
2. Implement classes.

1 Merging containers

You are to implement a function called ECMergeContainers, which takes two parameters: (i) a list (container) of some lists (containers), and (ii) a container for output. Here, container is a STL container e.g., vector, set, etc. ECMergeContainers will merge all the lists of containers into a single container (which should be sorted).

1. Your code must be as flexible as possible. That is, you should allow different kinds of STL containers to be the input and output.
2. Your code must be concise: use no more than **10** lines of code inside the function!
3. For output, some containers allow duplicates and you should keep the duplicate. If the output container doesn't allow duplicate (like set), then it is OK to discard duplicate.

2 Polynomial

You are to implement a class called ECPolynomial. This class supports common operations for a polnomial of a single variable x. In particular,

1. Scale: multiply a constant to the polynomial.
2. +: add two polynomials.
3. *: multiply two polynomials.
4. Long division: calcualte the quotient and remainder when one polynomial is divided by another polynomial. Check Wikipedia page for long division of polynomials: `https://en.wikipedia.org/wiki/Polynomial_long_division`. You need to support two operators: / and %.

Refer to the provided starter code (the header file) for the exact signatures of the interface functions.

3 Rational

In this assignment, you need to implement a class called ECRational, supporting generic rational of two quantities. The quantities can be C++ value types (e.g. integers, floating point) or user-defined value types (e.g. complex number, or polynomials). You may assume the data type supports the following: (i) default constructor and copy constructor, (ii) assignment operator, and (iii) arithmetic operators: +, -, * and / (division). Note: you don't need to simplify the rational. That is, it is OK to have common factors in numerator and denominator. For example, 4/4 = 1/1 are both acceptable. Therefore, ECRational should be a template class, with the following functions:

1. Customer constructor: taking two quantities as numerator and denominator.
2. Copy constructor, assignment operator,
3. +: add two rationals.
4. -: subtract two rationals.
5. *: multiply two rationals
6. /: divide two rationals.
7. Two methods to return the values of numerator/denominator.

Operator overloading

Please note that you should overload the operators to allow users to use your code as flexible as possible. For example, you should support the following (we omit the template for clarity):

ECRational x = 1 + ECRational(2,3);

Here, $x = 5/3$. How would the above code work? C++ would first invoke the custom constructor on the single integer 1 to construct an *ECRational* and then it invokes the + operator.

To do this, you should use the "friend" way for operator overloading.