Homework 2 — Counter<T>

Due Sunday, October 6th at 11:59pm

(50 points)

Objectives:

- Implement generalized c++ functions/classes
- Use "mini" c++ topics that we have covered: const
- Design and implement unit tests for a templated class

Credit:

- hw2.zip, containing Counter.[h|hpp], test.cpp, Makefile, TestTypes.[h|hpp], TestTypes.[cc|cpp], and main.cpp as appropriate. You do not need to include catch.hpp in your zip file.

Instructions:

You may complete this assignment with a partner if you so choose. If you work with a partner, include comments in test.cpp indicating which part(s) which partner did.

Your job is to implement a templated Counter class in C++. A Counter is a specialized type of map (dictionary) that counts the occurrences of hashable objects. You can think of it as a version of a std::map<T, int> with a fancy interface. For our Counter<T>, counts are allowed to be any positive integer value or 0.

If you find writing a main.cpp helpful, you may do so but this is not required.

Your Counter<T> class must provide the following functionality:

○ increment the count of an object T by one
○ increment the count of an object T by an amount n
○ decrement the count of an object T by one
○ decrement the count of an object T by an amount n
○ access the n most/least commonly occurring objects of type T
○ access normalized weights for all objects of type T seen so far
  - normalized weights means that each value of type T would be associated with the percentage of all items of type T that have been counted that are that value.
  - it essentially converts each T, int pair to a T, double pair where the double is the percentage rather than the raw count
○ access the set of all keys in the Counter
○ access the collection of all values in the Counter
○ access a converted Counter as a "regular" c++ map
○ access the total of all counts so far (how many objects have been counted)
○ access the total of all counts for objects T given a certain range (a minimum T and a maximum T element)
○ remove an object T from the Counter
○ access the count associated with any object T, even for values of T that have not been counted
○ initialize an empty Counter<T>
○ initialize a Counter<T> appropriately from a vector or array that contains type T
○ overload the << operator

Your Counter<T> must work for types T that are new, custom types, such as programmer-defined structs and classes. Each method that you implement must be adequately tested. You do not need to test each method with a Counter<T> of every type that T could be (that would be impossible!), but your different TEST_CASEs should make use of Counters that hold a variety of different types.

You must define one custom enum (such as your SquareType), one custom struct (such as the Book we've used in lecture), and one custom class (such as your Point, Rectangle, etc) with which you test your Counter<T> in addition to testing it with primitive types. Feel free to use enums, structs, and objects that you've already interacted with in this class to test your Counter<T>. Put these custom types in TestTypes.[h|hpp], TestTypes.[cc|cpp] as appropriate. You should have at least one TEST_CASE that uses a Counter containing each of these types.

See examples in the examples folder on github for how to write templated classes and functions, as well as the resources linked to in the resources document.

We are happy to clarify any methods/requirements that you'd like guidance on, so please, make sure to ask if you have any questions.

As always, your functions should be well documented. Since a main.cpp is not required, include your file comment with your name(s) and instructions for running your program in test.cpp.