

Homework 2 — Counter<T>

Monday, Oct 11th, 11:59pm - no extension applicable
(50 points)

Objectives:

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

Turn in:

- Counter.hpp, test.cpp, Makefile. You are not required to turn in main.cpp though you are highly encouraged to write a main as you test your Counter object! You do not need to turn in catch.hpp.

Instructions:

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 do not have experience working with c++ maps, see the end of this write-up for examples to get started.

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 interface:

<u>Function Signatures</u> Note: it is your job to determine which parameters and methods should be const!	<u>Description of behavior</u>
Counter(); Counter(std::vector<T> vals);	initialize an empty Counter<T> initialize a Counter<T> appropriately from a vector or array that contains type T
int Count(); int Count(T key);	access the total of all counts so far access the count associated with any object T, even for values of T that have not been counted
void Remove(T key)	remove the object T from the Counter
void Increment(T key); void Increment(T key, int n);	increment the count of an object T by one increment the count of an object T by n
void Decrement(T key); void Decrement(T key, int n);	decrement the count of an object T by one decrement the count of an object T by n
std::set<T> Keys();	access the set of all keys in the Counter



<code>std::vector<int> Values();</code>	access the collection of all values in the Counter
<code>std::ostream& operator<<(std::ostream& os, const Counter<U> &b);</code>	<p>overload the << operator for Counter<T></p> <p>This should print out the contents of the Counter in the format:</p> <p>{T: count, T: count, T: count, ..., T:count}</p>

Counter<T> and different types:

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.

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.

Some thoughts on getting started:

Though you may have the inclination to start by writing a non-templated version of your Counter and then converting it, our experience has been that getting a templated class started in c++ can be difficult enough that this might make finding your compiler issues harder. Therefore, we recommend the following steps:

- 1) Define your Counter<T> class with just a constructor.
- 2) Make sure you can create a Counter<int> (or some other primitive/built in type).
- 3) Write unit tests for one of the Counter<T> methods
- 4) Implement the Counter<T> method
- 5) Run your tests
- 6) Go back to step 3 and repeat until complete

Rubric Outline

Counter<T>	<ul style="list-style-type: none"> - these will be roughly equally spread between all methods that we've asked you to implement 	45 points total
Unit tests	<ul style="list-style-type: none"> - TEST_CASEs and SECTIONs used appropriately - each method appropriately tested <ul style="list-style-type: none"> - Note: no unit testing required for overloading the << operator 	20 points



Style and comments	<ul style="list-style-type: none"> - const and overloading used appropriately (5 points) - follows style guidelines (2.5 points) - commented appropriately (2.5 points) 	10 points
--------------------	--	-----------

Using Maps in C++

A map is an associative array. It links unique keys to values. You can imagine it as a vector except instead of having integer indices from 0 to the vector's size - 1, the "indices" can be of whatever type you want.

```
std::map<std::string, double> words_to_numbers;
// adding elements one by one
words_to_numbers["cat"] = 3.5;
words_to_numbers["dog"] = 5.2;
words_to_numbers["mouse"] = -100.0;
// updating a value
words_to_numbers["mouse"] += 5;

// getting the value associated with a key
std::cout << words_to_numbers.at("mouse") << std::endl;

// creating a map with values
// std::map<int, bool> ints_seen{{1: true, 2: true, 5: false}};

// iterating through maps
// option 1: with an iterator directly
std::map<std::string, double> :: iterator it;
for (it = words_to_numbers.begin(); it != words_to_numbers.end(); it++) {
    // access the key with it->first
    // access the value with it->second
}
// option 2: with a "for each" loop
for (std::pair<std::string, double> pair : words_to_numbers) {
    // access the key with pair.first
    // access the value with pair.second
}

// testing to see if an element exists in a map
if (words_to_numbers.find("cat") != words_to_numbers.end()) {
    // this value exists in this map!
    std::cout << "found cat!" << std::endl;
}

// using the insert method to insert a pair
// empty map container
```



```
std::map<int, int> num_to_num;

// insert elements in random order
num_to_num.insert(std::pair<int, int>(1, 40));
num_to_num.insert(std::pair<int, int>(5, 30));
num_to_num.insert(std::pair<int, int>(3, 40));
num_to_num.insert(std::pair<int, int>(4, 20));

// erase an element
num_to_num.erase(num_to_num.find(5));
```

