Homework 7: Templatized Dynamic Bag

Due 11:59pm Wednesday, November 12, 2025

CSCI 60

Krehbiel

Overview.

This program will have you write just a few functions to finish our implementation of a templatized Bag class that uses dynamic memory in a manner representative of the *vector* class in the Standard Template Library. Download the *bag.h* starter file, and put all your implementation in that file; this will be the only file you upload. You should also download the empty *bag.cpp* and *bagmain.cpp* from our Week 7 lectures so that you can test the code that you write; *bag.cpp* should stay empty, and you should use *bagmain.cpp* frequently as you write your code, but you will not upload it, so you can use it however you like.

Milestone 1: Finish our implementation of erase_one.

Our implementation from class (which is in the starter file) appears to have the correct functionality when we test it in, but it fails to maintain the invariant that a well-formed bag object should always be at most 100% and *greater than 25%* full relative to the capacity of the array that has been dynamically allocated for it. If *erase_one* decreases the size from

0.25×cap to <=0.25×cap, you should deep copy the contents of the array (at some sensible point in your implementation!) to a new array with half the capacity of the bag when the function was called.

You should write test code to make sure that your modifications don't create a bug in the contents of a bag before and after an *erase_one* call, but you'll have to rely on your correct memory diagrams to make sure that you are establishing the invariant about size relative to capacity. As with the other functions, you should make sure this function does not orphan any memory.

Milestone 2: Implement erase.

The precondition of *erase* should be that the bag is well-formed, and the post-condition should be that the bag is well-formed with all occurrences of the target value removed and with the number of removed elements returned. This should be very straightforward if you use *erase_one* as we did in our previous bag implementation!

Milestone 3: Implement erase_fast.

The pre- and post-conditions of *erase_fast* are the same as those of *erase*, but this implementation will have to be more clever than the one inspired by our previous implementation. In particular, you need to write a *linear time* algorithm, and you may only need only one pass of the bag's array at most twice for each call to *erase_fast*. Hint: You'll want to use one of these passes to erase the elements, and once to reallocate and populate a smaller array if needed.

Milestone 4: Overload += as a member function.

The pre-conditions for *+=* are that the calling object and righthand operand are both well-formed bags, and the post-condition is that the calling object has all of the contents of the righthand operand appended to the end. You may implement this however you are able to achieve these post-conditions!

Finishing touches:

As always, before you submit make sure to remove code from debugging, and generally review the style guidelines from HW1. Add the pre- and post-conditions mentioned in this handout as your function comments, and add additional comments to the bodies of your functions where it might help orient the reader.