Project 2: Templates, Pointers, Arrays, and Linked-Lists

Dr. Oyewole Oyekoya

Due March 5, 2020 by 5PM

Implementation:

In Project 2, we will work with the LinkedBag class discussed during lecture. This project consists following five modifications to the LinkedBag class:

1. Revise the public **add()** method in the class LinkedBag so that the new node is inserted at the end of the linked chain.
2. Create the method
   void deleteSecondNode()
   which has the following functionality: If headPtr is a pointer variable that points to the first node of a linked chain, the **deleteSecondNode()** method will delete the second node of the chain.
3. Suppose that the class LinkedBag did not have the data member itemCount.
   - Revise the method **getCurrentSize()** so that it counts the number of nodes iteratively.
   - Create the method
     int getCurrentSizeRecursive()
     which counts and returns the number of nodes in the chain recursively.
4. Add the parameterized constructor:
   LinkedBag(ItemType entries[], int entryCount)
   which creates a bag from a given array of entries.
5. Create the method
   ItemType removeRandom()
   that removes an entry from the bag in a pseudo-random fashion and returns it to the system.

Testing

You must always implement and test you programs **INCREMENTALLY!!!**

What does this mean?

- Implement and test one method at a time.
- For each class:
  1. Implement one function/method and test it thoroughly (multiple test cases + edge cases if applicable)
  2. Implement the next function/method and test in the same fashion.

How do you do this?

Write your own **main()** function to test your classes. In this course you will never submit your test program, but you must always write one to test your classes. Choose the order in which you implement your methods so that you can test incrementally (i.e. implement mutator functions before accessor functions). Sometimes functions depend on one another. If you need to use a function you have not yet implemented, you can use **stubs**: a dummy implementation that always returns a single value for testing (don't forget to go back and implement the stub!!! If you put the word STUB in a comment, some editors will make it more visible

Grading Rubric:

- **Correctness 80%** (distributed across unit testing of your submission)
  - A submission that implements all required classes and/or functions but does not compile will receive 40 points total (including documentation and design).
- **Documentation 10%**
- **Style and Design 10%** (proper naming, modularity, and organization)

Submission:

You will submit **the following files:**

- LinkedBag.cpp
- LinkedBag.h
- BagInterface.h

Your project must be submitted on Gradescope.
Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You MUST test and debug your program locally. Before submitting to Gradescope you MUST ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope, and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope. "But it ran on my machine!" is not a valid argument for a submission that does not compile. Once you have done all the above you submit it to Gradescope.