**Approach to PROJ 1, Bag ADT**

Riley Dorough / Chandler Whitley

27JAN20

This project was completed through the collaborative efforts of the ultra-genius Chandler Whitley & the innovative Riley Dorough.

**Bag**

Requirements & Contemplation:

* Fills like a stack (FILO), but searching begins at the last (newest) element.
* Requires access any item in the Bag ADT.
* Main passes Dictionary\_Size{10} to the *BDictionary* class, then to the *ABag* class, which creates an object of class *ABag* with the given value’s size.

**ABag.h**

To attack this project, first map the dependency of the headers from the ground up and find the lowest level header that allows us to edit it. This should be the *ABag.h* header. This header should include our implementation of a *BagADT* and include *BagADT* related variables & methods.

ABag variables:

* Maximum capacity
  + Private variable for the max capacity of bag.
* Current position
  + The current logical size of the bag.
* Pointer of type <E> to access elements of the *BagADT.*

ABag class (construct & destruct):

* Constructor
  + Maximum capacity should be set to equal the value provided from the *BDictionary* header.
  + Then call the empty bag method to ensure a fresh start.
  + Use polymorphism to assign a new array size to each object created.
* Destructor
  + Delete the object to free up memory and prevent leakage.

ABag methods:

* Add Item
  + Inside the BagADT first check to ensure that the bag is not full, if the bad is not full insert an item into the current position then increment the position by up by 1.
* Remove
  + First the remove method runs the find algorithm to check if the item is in the bag. Then when the find algorithm finds the selected value, it sets a temporary variable to the position of that value in the bag. Next, another for loop will start at the temporary variable’s position and shuffle every item down by 1 position. Finally, the position variable is subtracted by 1. Duplicated variables are ignored since the bag position variable will not allow the user to access the left over data and the maximum size will never allow for memory leak.
* Remove Top
  + First check to ensure that the position is not equal to 0 if it is, return false. If the bag contains values then update the return value to the value at the current position then decrement the bag position variable by 1 and return true.
* Find
  + Start at the tail of the bag and search using a top down approach. I.E. a decrementing for loop. Since the capacity is a constant integer of 10 this is a feasible search method. Find will start checking at the bag position and decrement down to position 0 if the value searched for does not exist the method returns false.
* Inspect Top
  + First check to ensure that the position is not equal to 0 if it is, return false. If the bag contains values then update the return value to the value at the current position and return true.
* Empty Bag
  + Set the position variable to 0 and ignore all the sad little left over values. Since we are using smart variables the memory will deallocate when the program terminates.
* Operator+=
  + Makes a call to the add item method with the Rvalue being overloaded and added to the object.
* Size
  + Returns the bag position variable.
* Bag Capacity
  + Returns the constant integer declared in *bagtestmain.cpp*.

**BDictionary.h**

Handles the “management” of what methods to call on with the given data and KVPair. This should help hide implementation from the end user.

BDictionary variables:

* Pointer to the object of class *ABag* with type *KVPair*

BDictionary class (construct & destruct):

* Constructor
  + Should use polymorphism to reinitialize a private data member to the size that is passed into the BDictionary constructor from the main .cpp file.
* Destructor
  + should delete any private data members.

BDictionary methods:

* Clear
  + Calls on the empty bag method from ABag header
* Insert
  + Create a KVPair object based off of the two variables given to this method from main.cpp.
  + This new KVPair is passed to the add item method.
* Remove
  + Passes in a key to create a temporary KVPair object and then calls upon the removed function in the ABag header.
  + Updates the return value with what pair was removed.
* Remove any
  + Uses a KVPair object without a key and then calls upon remove top.
  + Updates the return value with what pair was removed.
* Find
  + Create a KVPair object based off of the two variables given to this method from main.cpp.
  + This new KVPair is passed to the find method in ABag header.
  + Updates return value with desired value.
* Size
  + Calls the size method in ABag header.

**BagTestMain.cpp**

1. Assign objects of class BDictionary to initialize as smart pointers.

2. Enjoy coffee knowing that we finished this project.

3. Get an A because we are a great team!

**Coding Process**

Step 1 (day 1): Read and Learn

* Read through the instructions and guidelines and form a rough plan of action
* Research smart pointers
* Review polymorphism because I think it will be used in this lab
* Describe each function for each class
* Decide which private data members to use

Step 2 (day 2): Pseudocode ABag

* Define data members
* Write each function in pseudocode

Step 3 (days 3-4): Program ABag

* Create a project with appropriate provided files
* Make new ABag.h file for a class that inherits from bagADT
* Code each function from bagADT in c++ language
* Discuss with teamate to make sure implementation makes logical and syntactical sense.
* Go to Dr. Sipantzi’s office and check that I am on the right track

Step 4 (days 3-4): Test ABag

* Write main program to test ABag function for the correct implementation

Step 5 (day 5): Pseudocode BDictionary

* Define data members for BDictionary and think about how it will interface with ABag
* Write BDictionary functions in pseudocode off of ABag functions

Step 6 (days 5-7): Program BDictionary

* Make BDictionary inherit DictionaryADT
* Code each function in c++
* Discuss with teammates and test
* Go to office hours as necessary to work out any bugs / errors and to check logic structure

Step 7 (days 7-12): Finish product

* Go through directions again and ensure all instructions were followed
* Make any necessary changes to follow rules
* Ensure all functions are used for maximum credit
* Clean up code and add necessary clarifying comments
* Create a finished Zipped packet and turn in