Programming Assignment 1 –   
Bag-based Dictionary

Implement a dictionary using a Bag—Project 4.7 in the text (modified)

Use the bag ADT provided to create an array or vector-based implementation for bags. Then use your bag to implement the dictionary ADT provided you. To get full credit for the assignment you must incorporate one or more smart pointers and tell me where they are in your Approach document – if you make me look for them you won’t get credit for them. If you don’t know what smart pointers are then you’ll need to do some research.

Test your bag and dictionary implementations with the bagtestmain.cpp file I’ve provided for you. If you are not able to fully implement all the functions of the dictionary and/or bag, you may modify the tests in bagtestmain.cpp to only exercise those functions you were able to complete.

You will only get credit for those methods you test and display, so be sure you don’t leave any out.

Also, you cannot add any public functions to your bag and dictionary implementations beyond those specified in the ADTs.

Put the following files into a zip file named student\_name\_Bag\_Assignment and submit them to Blackboard:

* ABag.h // Your bag-array implementation which must inherit the Bag class
* BDictionary.h // Your Dictionary-bag implementation which must inherit the // Dictionary class
* bagtestmain.cpp // The test driver I have provided for you
* bagADT.h // The bag ADT I gave you – it should be unchanged
* dictionaryADT.h // The dictionaryADT I gave you – it should be unchanged
* kvpair.h // The kvpair class I gave you – it should be unchanged
* screen\_shots.doc/txt // Word document with screen shot(s) showing   
   // program in action or a text file showing the test   
   // program's output.
* Approach.doc // This is a Word document where you explain how you implemented  
   // your solution and how you tested each required function.
* Your\_dictionary.exe //Your executable file.\*\*\*

Note: If your ABag does not inherit Bag and/or BDictionary does not inherit Dictionary, you will not receive any credit for your work.

\*\*\* If you completed your assignment using Visual Studios you must use Visual Studios 2017 or 2019 and I would like you to submit your entire VS project directory.

Your test program must exercise every function of the dictionary. For any function whose functionality is not obvious you must explain in your Word document how your test output demonstrates that function. See me if you have questions.

See Blackboard for the assignment due date. **Late assignments will not be accepted.**

Rubrics:

* **Program must run in order to get any points. By “run” I mean that at you must at least get one or more of the bag methods working (and your program must demonstrate that functionality).**

Tips for Success

Start by working on your “Approach” first. Once you are satisfied with your approach, then start building your program incrementally. Start with the bag and increment one feature at a time (you’ll have to stub out the features the ADT requires that you are not ready to implement yet) starting with the constructors and then working your way down the feature list using common sense to figure out which features need to be implemented first. Try your bag out with the various parameter combinations I want you to test with (<int, string> and <string, int>). When you are satisfied the bag is working then move on to the dictionary, again implementing and testing function by function.

Don’t wait until the last minute. You’ll find that many of your problems you will solve while you are away from your computer and have a chance to think about the error you are seeing. This takes time.

Note: KVpair, which uses the == operator for comparing the key values, will only accept objects that have also implemented the == operator. This class has been tested with the following types:

* + - string
    - int

It specifically does not work with the Int type (at least not in the version of C++ I am working with).

## Debugging your code

A big part of this assignment is debugging your code, so do not expect your instructor to do this for you. Having completed the pre-requisite courses for this class, you should already have some experience finding coding errors and fixing them. This class will give you plenty of opportunities to further refine those skills and you can only do that by **wrestling with the problem**. Here are some debugging tips:

* Build a little, test a little. Don’t write all your code before you start debugging it. Break your work into logical chunks that can be compiled and debugged and build them one at a time. For this project, I would start by building the Bag class and implementing the addItem() function first. Once I get that function working properly, then I would move on to another Bag function. The idea is you build and test a function one function at a time. That way, if you run into an error, you know where to look.
* Learn to use the debugger if you haven’t already. The debugger allows you to step through your code one step and a time and see what happens in memory while you’re doing it. When students come to me with problems, I first try to isolate where the problem is logically based on what the program is doing and then I use the debugger to find the actual fault. Here is an excellent video tutorial on the Visual Studios debugger: [How to DEBUG C++ in VISUAL STUDIO](https://youtu.be/0ebzPwixrJA).
* Be willing to walk away from your computer and give your mind a rest. I find that the solution to a problem often comes to me while I am doing something else. I have never regretted stepping away from my computer to let the problem “percolate” in my mind, but I have often regretted not doing this. This is one of the reasons you should never wait till the last minute to start working on your program; by doing that you are not giving yourself the time to walk away.