Assignment 3 – Spell Checker

Due Date: 11:59 pm on October 3, Friday

Objectives:

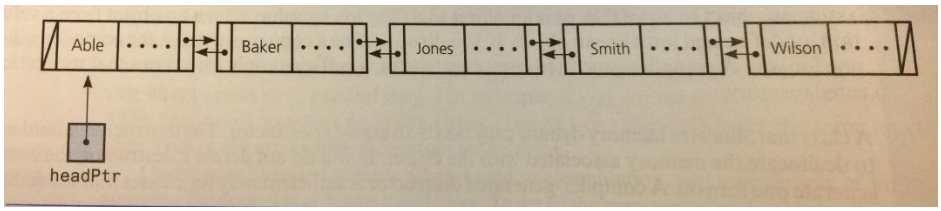
- Students will gain an experience to use doubly-linked list to solve a problem.

Problem Descriptions:

This project consists of two parts. The first part is to write a **class** that implements the ADT Bag using a **doubly-linked list** with a pointer to its first node. The second part of the project specifies how a program will use the class.

Part I

In a doubly linked list, each node can point to the previous node as well as to the next node. Figure 4-10 shows a doubly linked list and its head pointer. Define a class **DoublyLinkedBag** that implements the ADT bag using a doubly linked list as shown in Figure 4-10.



Requirements

1. First define a class to represent a node in a doubly linked list by modifying the node class we used for linked base implementation of ADT Bag. Test it before you use it for the class **DoublyLinkedBag.** This node class should have the following data members:
   - a pointer to the next node
   - a pointer to the previous node
   - an item
2. Define and implement class **DoublyLinkedBag** following the example of the **LinkedBag** we discussed in the class.
   a. To be able to run your program on **GitHub**, you must include the class implementation file in the header file `DoubleLinkedBag.h` as below:

      .....
      `#include "DoublyLinkedBag.cpp"`
      `#endif`
   b. When implementing the "add" member function, you need to consider the case that the link is empty. Otherwise core dump.
   c. To be able to run your program on **GitHub**, you must include the class implementation file in the header file `DoubleLinkedBag.h` as below:

      .....

```
#include "DoublyLinkedBag.cpp"
#endif
```

3. Use the following program to test your ADT Bag implemented by doubly linked list

```
// This program tests the ADT Bag class which is implemented by a doubly
// linked list.

#include <iostream>
#include <string>
#include "DoublyLinkedBag.h" //doubly linked list bag

using namespace std;

void displayBag(const DoublyLinkedBag<string>& bag)
{
       cout << "The bag contains " << bag.getCurrentSize()
   << " items:" << endl;
   vector<string> bagItems = bag.toVector();

   int numberOfEntries = (int) bagItems.size();
   for (int i = 0; i < numberOfEntries; i++)
   {
      cout << bagItems[i] << " ";
   }  // end for
       cout << endl << endl;
}  // end displayBag

void bagTester(DoublyLinkedBag<string>& bag)
{
   cout << "isEmpty: returns " << bag.isEmpty()
   << "; should be 1 (true)" << endl;
   displayBag(bag);

       string items[] = {"one", "two", "three", "four", "five", "one"};
   cout << "Add 6 items to the bag: " << endl;
       for (int i = 0; i < 6; i++)
   {
              bag.add(items[i]);
       }  // end for

   displayBag(bag);

   cout << "isEmpty: returns " << bag.isEmpty()
   << "; should be 0 (false)" << endl;

   cout << "getCurrentSize: returns " << bag.getCurrentSize()
   << "; should be 6" << endl;

   cout << "Try to add another entry: add(\"extra\") returns "
   << bag.add("extra") << endl;

   cout << "contains(\"three\"): returns " << bag.contains("three")
   << "; should be 1 (true)" << endl;
   cout << "contains(\"ten\"): returns " << bag.contains("ten")
   << "; should be 0 (false)" << endl;
   cout << "getFrequencyOf(\"one\"): returns "
   << bag.getFrequencyOf("one") << " should be 2" << endl;
```

```
   cout << "remove(\"one\"): returns " << bag.remove("one")
   << "; should be 1 (true)" << endl;
   cout << "getFrequencyOf(\"one\"): returns "
   << bag.getFrequencyOf("one") << " should be 1" << endl;
   cout << "remove(\"one\"): returns " << bag.remove("one")
   << "; should be 1 (true)" << endl;
   cout << "remove(\"one\"): returns " << bag.remove("one")
   << "; should be 0 (false)" << endl;
   cout << endl;

   displayBag(bag);

   cout << "After clearing the bag, ";
   bag.clear();

   cout << "isEmpty: returns " << bag.isEmpty()
   << "; should be 1 (true)" << endl;
}  // end bagTester

int main()
{
        DoublyLinkedBag<string> bag;
        cout << "Testing the Link-Based Bag:" << endl;
   cout << "The initial bag is empty." << endl;
        bagTester(bag);
   cout << "All done!" << endl;

   return 0;
}  // end main
```

Part II

Consider spell checking problem: a program reads a word and checks if it is spelled correctly. You can use a bag to serve as a dictionary that contains a collection of correctly spelled words. To check if a word is spelled correctly, you check whether it is contained in the Bag that serves as a dictionary. Use this scheme to create a spell checker for the words in an external file. To simplify your task, restrict your dictionary to a manageable size.

[Hint] Please note that C++ string treats two strings as different even if they are same strings except for upper or lower cases. One way to perform checking correctly, don't include words with any upper cases in your dictionary file . When checking each word in a file, first convert it to all lower-case characters using C++ library function **tolower()** and then see if it is in the dictionary.

Requirements

- Use DoublyLinkedBag in Part I.
- Create the dictionary using a list of correctly spelled words in a file.

Input

- The name of an external file that contains words to check.

Output

- A list of incorrectly spelled words in the input file.

An Example of Test Runs on GitHub

The output of your program might look like this:

```
>g++ -std=gnu++0x project3.cpp
>./a.out
      Enter the name of the file that contains words to check:
      myreport.txt

      The following words in the file "myreport.txt" are not
      spelled correctly:
      Stude
      Reseach
      Resul
      Outpt

      Thanks for using the spell checker system.
```

What to Hand In

- Submit and test all source programs of Part I and II to your class account in **GitHub** system.
- Submit the following documents to the drop box **Project3** on D2L:
  - all source programs and the following two files
    - the text file as a dictionary file
    - the text file for spelling check
  - a script file for test runs for Part II
  - a word file for Part II that contains
    - design document
    - test data with an explanation
    - user document

    (Please follow the details of the document About Programming Assignments posted on D2L.)

Grading

Requirements	points
Comments in the program	10
Program correctness for Part I	20
Program correctness for Part II	20
Script file for several test runs for Part II	20
Design document for Part II	10
Test data with explanation	10
User document	10
TOTAL POINTS	**100**

No Submission Made!!!!!!

Requirements	points
Comments in the program	0
Program correctness for Part I	0
Program correctness for Part II	0
Script file for several test runs for Part II	0
Design document for Part II	0
Test data with explanation	0
User document	0
TOTAL POINTS	**0**