CSCI 235 - Spring 2015
Homework 2

Due: Sunday, March 15th, 2015, 11:59 pm

Follow the instructions presented in the **Programming Rules** document (in the Course Information folder). Submit only the header and source files as well as your Makefile (I will cover this in class). Note that your program needs to compile in order to get any credit at all. Adhere to the style guidelines. Start Early!

Note that you have to implement some functions **using recursion**. Chapter 3.3 describes in detail how recursion is used to implement linked list methods, please work through the examples before starting this homework.

1 Polynomials

Modify the LinkedBag implementation in order to implement operations on polynomials of one parameter. For example the polynomial:

$$3x^7 + 4.1x^5 + 7x^3 + 9x^0$$

can be represented as:

```
head_ptr_ -> [3, 7] -> [4.1, 5] -> [7, 3] -> [9, 0] -> nullptr
```

The degree of the above polynomial is 7.

In order to achieve this you need to do the following:

1. Modify the Node class so that it now holds two items: `coefficient_` and `exponent_`. The first node in the above example has `coefficient_` = 3 and `exponent_` = 7. That means that instead of `GetItem()` you will have `GetCoefficient()` and `GetExponent()`, instead of `SetItem()` you will have `SetCoefficient()` and `SetExponent()`, etc. Modify all other functions accordingly.

2. You will also need to modify the LinkedBag class in various ways. You can now call it LinkedPolynomial. Also, there is no need to use the BagInterface.h class.

   (a) Remove the functions `ToVector()` and `Remove()` from the LinkedPolynomial implementation.

(b) Modify the `Add()` function so that it **now adds at the end of the list**. Note, that now the `Add()` will take two parameters, a coefficient and an exponent. Also `Add()` should not add a node with an exponent that is already there. So for example the following code:

```
LinkedPolynomial<double> polynomial;
polynomial.Add(3, 7);
polynomial.Add(4.1, 5);
polynomial.Add(7, 3);
polynomial.Add(8.1, 5);
```

Should result to

```
head_ptr_ -> [3, 7] -> [4.1, 5] -> [7, 3] -> nullptr
```

The last Add() did not have any effect because a node with exponent 5 was already there.

You don't have to use recursion for the `Add()` function, but if you want, you may.

(c) Add a member function called `DisplayPolynomial()` that will traverse the list **recursively** and will `cout` the coefficients and exponents. Note that this is a `const` function. Display in the following form:

```
3 * x^7 + 4.1 * x^5 + 7 * x^3 + 9 * x^0
```

(d) Add a member function called `Degree()` that returns the degree of the polynomial (or 1 if the polynomial is empty). Note that this is a `const` function. In the previous example `polynomial.Degree()` should return 7.

(e) Add a member function called `ItemType Coefficient(const ItemType& exponent)` that will return the coefficient of a given exponent. For example `polynomial.Coefficient(5)` should return 4.1. Note that this is a `const` function. You have to implement this using **recursion**. Note that if a term of a certain exponent is not seen in the polynomial, then it's coefficient is 0.

(f) Add a member function called `bool ChangeCoefficient(ItemType new_coefficient, ItemType exponent)` that changes a coefficient for a given exponent. For example if you call `polynomial.ChangeCoefficient(100, 3)` the resulting polynomial will change to

```
head_ptr_ -> [3, 7] -> [4.1, 5] -> [100, 3] -> [9, 0] -> nullptr
```

You have to implement this using **recursion**.

In order to test the above do the following:

1. Write a `client` function (you can place it on top of `main()`) `Polynomial<double> CreatePolynomialFromInput()` that prompts the user to provide a sequence of coefficients/exponents and then uses the `Add()` member function to add them to the Polynomial.

2. Then write a client function called Test Polynomial() that first calls `CreatePolymomialFromInput()` to generate a new polynomial and then does the following in sequence:

   Calls the `DisplayPolymomial()` function.

   `couts` the `Degree()` of the Polymomial.

   Asks the user to provide an exponent.

   `couts` the Coefficient(exponent).

   Asks the user for a new coefficient (call it new_coefficient).

   Calls the functions `ChangeCoefficient(new_coefficient, exponent)` and `couts` its return value (either true of false).

   Calls the `DisplayPolynomial()` function.

Finally, write a member function to add a given polynomial to the current one: `void AddPolynomial(const LinkedPolynomial<ItemType> &b_polynomial)`. So if the `b_polynomial` is

```
head_ptr_ -> [1, 8] -> [2, 5] -> [8, 0] -> nullptr
```

Then the current polynomial will change to:

```
head_ptr_ -> [1, 8] -> [3, 7] -> [6.1, 5] -> [7, 3] -> [1, 0] -> nullptr
```

In order to test the above write a function called TestAddition() that does the following: Calls the function `CreatePolynomialFromInput()` twice to generate two polynomials `polynomial1` and `polynomial2`. Adds the second one to the first one (i.e. calls `polynomial1.AddPolynomials(polynomial2)`). And finally displays the result by calling `polynomial1.DisplayPolynomial()`.

2 Submission

Once you have ensured that your program works on the Linux machines in Lab 1000G in Hunter North, pack all files in a zip/tar.gz archive and name it `<YOUR_FULL_NAME>_hw2.zip` or `<YOUR_FULL_NAME>_hw2.tar.gz`. Replace `<YOUR_FULL_NAME>` with your full name, using underscores for spaces.

Submit this file on Blackboard in the section for Homework 1.

Please make sure you have included the Makefile and that your program compiles using the `make all` command. Your program will not be graded at all if any of the following is true:

- The files were not archived (zipped)
- There is no Makefile
- There is a compile error
- It's missing any of the files required by this assignment

Again, please read the Programming Rules document on Blackboard.

Good luck!