Programming Assignment 2

CSCI 235 section 2
Instructor Silvano Bernabel
Due Friday October 7 (11:59 PM EST)

Follow the instructions presented in the Programming Rules document. Submit the header and source files, as well as your Makefile. Note, your program needs to compile in order to get any credit.

Begin by reading the following files: *Node.h, Node.cpp, LinkedBag.h, LinkedBag.cpp.* With some modifications on these files you will be implementing an ADT for arbitrary size integers in base 10. You will represent an integer in the following way:

```
head_ptr_->[3,5]->[0,4]->[0,3]->[1,2]->[8,1]->[9,0]->nullptr
```

which is the integer 300189 in decimal.

1. Modify the Node class so that it contains two items: `coefficient_` and `exponent_`. The first node in the example given above has `coefficient_=3` and `exponent_=5`. As a result instead of having the method `GetItem()`, you will have `GetCoefficient()` and `GetExponent()`. Similarly, instead of `SetItem()`, you will have `SetCoefficient()` and `SetExponent()`. The remaining methods that use the original functions need to be changed accordingly.

2. Modify the LinkedBag class in the following ways:

   (a) Rename the LinkedBag class to BigInteger.

   (b) Delete functions `Remove()` and `ToVector()` from the BigInteger implementation.

   (c) Rename the `Add()` method to `Insert()`. Modify `Insert()` so that it inserts elements at the end of the list. This method will need to take two parameters, a coefficient and an exponent. The input range for the coefficient will be $0 \leq |c| \leq 9$ (where c is the coefficient). And the maximum exponent allowed must be 99. You will need to handle any out of range inputs. Additionally, you must not allow for the leading coefficient to equal zero. If an exponent is already present in the list, `Insert()` should not replace the coefficient. For example,

   ```
   BigInteger number;
   number.Insert(3,5);
   number.Insert(0,4);
   number.Insert(0,3);
   number.Insert(1,3);
   ```

   should yield,

   ```
   head_ptr_->[3,5]->[0,4]->[0,3]->nullptr
   ```

A node with exponent equal to 3 was already present, thus the coefficient does not change.

(d) Create a member function called `DisplayBigInteger() const` that will traverse the nodes and cout the decimal representation of the number. Note, you have to display the missing digits too.

(e) Create a member function called `int Coefficient(const int& exponent) const` that returns the coefficient of a given exponent.

(f) Create a member function called `bool ChangeCoefficient(int newCoefficient, int exponent)`. This function will return true if the coefficient present is equal to the input, thus no change was needed.

Testing the methods:

i. Create a client function called `BigInteger CreateBigIntegerFromInput()` You can place this function above the `main()`. This function will prompt the user to input a sequence of coefficients and exponents. It then uses the `Insert()` method to insert the values in the BigInteger.

ii. Write a client function called `TestBigInteger()` that calls `CreateBigIntegerFromInput()` to create a new BigInteger, then does the following in order:

Calls the `DisplayBigInteger()` function,
Asks the user to input an exponent,
Prints the `Coefficient(exponent)` for the given exponent,
Asks the user for a new coefficient,
Calls the `ChangeCoefficient(newcoefficient)` function and prints the return value,
Calls the `DisplayBigInteger()` function

3. Lastly, create a member function that will check equality of a new BigInteger to an existing BigInteger. The function will be called `bool EqualsBigInteger(const BigInteger& number) const`.

For example, given a BigInteger:

```
head_ptr_->[3,2]->[0,1]->[1,0]->nullptr
```

and checking equality to a new BigInteger:

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

the result should be `false`

Testing this method: Create a client function called `TestEquals()`. This function will do the following:

(a) Generate two BigIntegers with `CreateBigIntegerFromInput()`, number1 and number2.

(b) Check equality between number2 and number1 (i.e. `number1.EqualsBigInteger(number2)`).

(c) Display the result `true` or `false`.