Project 3: BigNumber
Due: - 3/25/2009

Objective
To gain additional experience working with the List class.

Problem
The maximum length of a long integer on some compilers is eight bytes (2^{63} -1). In some cases, the number's length is even more restricted (Visual C++ defines a long int at 4 bytes). Let's face it - this could be too short for some serious scientific applications. For this project, you will implement a BigNumber ADT (Abstract Data Type) that will be capable of handling arbitrarily big numbers.

Note: There is a program called "bc" (An arbitrary precision calculator language) which exists on Linux and Unix machines. This program can do calculations on super huge numbers. Type "man bc" at Linux command prompt for more information.

Create a BigNumber class using the STL List class.

Requirements
Your BigNumber class will need to include the following operations:

1. input string representing number using the *operator*>> (overload operator)
2. output list representing number using *operator*<< (overload operator)
3. overload *operator* == that returns true if equal and false otherwise
4. overload *operator* > that returns true if greater than and false otherwise
5. overload *operator* < that returns true if less than and false otherwise
6. overload *operator* + that returns the addition of 2 BigNumbers
7. overload *operator* - that returns the subtraction of 2 BigNumbers
8. overload *operator* = that assigns the value of BigNumber to BigNumber
9. *negation* (overload the unary - operator)
10. *copy constructor and destructor*

Functions (methods)

1. *length* - returns the numbers of digits
2. *absolute value* - this function returns the absolute value of a number.

Implementation Details
Create three files: BigNumber.cpp, BigNumber.h, and BigNumberTest.cpp

Example of input format of numbers
Integer -123456 can be passed in as "-000123456" or "-123456"
Integer 234567 can be passed in as "234567" or "000234567", but not "+000234567"
(In other words the number may contain leading zeros, but BigNumber will not contain those leading zeros)

Representation of BigNumber
Each BigNumber will be represented as a list. Each Node of the list should correspond to a single digit of the integer. Allowed digits are of base 10 (0,1,2,3,4,5,6,7,8,9). You may choose to store the least significant digit (one's column) at either the head or tail of the list:

Design
Spend considerable time designing your implementation before you sit down at the computer. Some design decisions (such as where to store the least significant digit) will have a considerable impact on how efficient your code is and how easy it is to program. Document any design decisions (tradeoffs) you make - these decisions should be included with your project writeup. Not all choices are explicitly listed in this assignment specification. How, for example, do you plan on indicating whether an integer is positive or negative?

Note: Your implementation of arithmetic operations should fundamentally consist of traversing linked lists. Do not use an array, string, or even an integer to dump your number into - in order to do the arithmetic operations. Also note that for the linked list representation of a big integer, it is not necessary to store (and you should not store) a field indicating how many nodes are in the linked list. [In fact it would be important not to store such a field - remember this is a BigNumber therefore theoretically the length of the number could exceed any integer container used to store its length.]

Hints and Comments

1. Ensure that your files are named BigNumber.cpp, BigNumber.h, and BigNumberTest.cpp. Create for yourself a BigNumberTest.cpp that will test important aspects of your program - comment each test so I can see what you are trying to test. A writeup of your test approach should be included with your project.
2. Hint for inputting Bignumbers - in order to do this you will need to overload the >> operator…this operator will use a built-in istream method called **get()**. You will have to input one character at a time and validate it against ASCII. Follow the rules for BigNumber representation given above.

Further hints:

```
void BigNumber::read(istream & in);  [within the class BigNumber]
istream & operator>>( istream& in, BigNumber & p );  [outside BigNumber]

The operator>>overload simply uses read to acquire a number from the
screen:

istream & operator>>( istream& in, BigNumber & p )
{
     p.read(in);
     return in;
}
```

How does the read function work?

```
      char ch;
      hold = NULL;
      while (legaldigit(ch=in.get(),firstneg,firstret,sign,flag))
      {
            if ((ch!='-') && flag==true)
            {
                  ntemp = new(nothrow) Digit(ch - 48);
.
.
.
```

legaldigit is a helper function for the class - its job is simply to check if the current entered "character" is legal.

```
//  Utility function for the read method
//
//  * flag - indicates a good character to store in our number
//  * first - indicates that we are still waiting for a first character
//  * The first part of the if statement below (digit == 10) is meant
// to burn off any return characters left over from previous input
//
bool legaldigit(char digit, bool& firstneg, bool& firstret, bool& sign,
bool& flag)
{
      if (((unsigned) digit==10 && firstret) || (digit >= '0' && digit
<= '9')  || ((digit == '-') && firstneg && !(sign==NEG)))
      {
            firstret = false; firstneg = false;
            if (digit=='-') { sign = NEG; }
            if (digit > '0' && digit <= '9') flag = true;
            if ((unsigned) digit==10) firstneg = true;
            return true;
      }
      else
      {
            return false;
      }
}
```

3. This maybe a good time to learn how to use a debugger – to see how to use the linux and unix (macintosh) debugger GDB go to: http://www.ibm.com/developerworks/library/l-gdb/
4. Freely place print statements throughout your program (rough draft only) that assists you in following how the program is progressing.
5. The addition overload is the meat of this project.
6. This program is relatively challenging - I would definitely not wait to the last moment to work on it.

```
BIG NUMBER DEMONSTRATION


Enter Value A: 111111111111111111111
Enter Value B: 22222222222222222222

Value A: 111111111111111111111
Value B: 22222222222222222222

  -A              -111111111111111111111
   A + B          3333333333333333333333
   A - B          -111111111111111111111
   A == B         false
   A != B         true
   A < B          true
   A >= B         false
   A > B          false
   A <= B         true


Assigning A to B
Value B :111111111111111111111

Enter new value A: -333333333333333333333
Value A: -333333333333333333333
Value B: 111111111111111111111
A == B: false
A < B: true

Use Copy Constructor to Create C using A
Value C: -333333333333333333333

Enter new value A: -444444444444444444444444
Value C: -333333333333333333333

Function called with A
Enter Value F1: 55555555555555555
Enter Value F2: 666666666666666666666666
Value D (A of Parent): -444444444444444444444444
Value D after D = F1+F2: 666666672222222222222221

Continue? ( Enter 'Y' to continue )
```