CSE240 Spring 2017 Project 5
Due time:Friday, March 31st, 5:00pm

1 Motivation:

The C++ long int can only take up to 64 bits of information, which may not be enough for some scientific computation. In this project, we will design and implement a class named UnsignedLargeInt which will overcome these problems.

The requirement for this project is to use UnsignedLargeInt at almost all situations when int is used.

2 Requirements:

As int, UnsignedLargeInt can be zero or positive. Unlike int, UnsignedLargeInt has "infinite" precision and can hold as many digits (char) as possible. The digits are represented by char, and are stored in a dynamically allocated array.

The size of the array can be larger than the actual number of digits in the number. For example, to add two UnsignedLargeInt a1 and a2, the resulted UnsignedLargeInt can have max(a1.len,a2.len) + 1 bytes allocated as the digit array, which is long enough to handle any overflow.

UnsignedLargeInt should have the following operators:

2.1 Constructors and destructor

There are 3 constructors:

- default constructor will set the UnsignedLargeInt to zero
- copy constructor to take another UnsignedLargeInt as the argument, because we are working with a dynamically allocated array, you must perform deep copy.
- constructor to take a unsigned long long as the argument, and transfer this unsigned long long as digits (char) and store the digits in the array. The argument can be positive or zero, so your UnsignedLargeInt must be able to reflect the sign of the argument.

- Destructor: C++ allows only one version of destructor, and it should delete the char array.

2.2 +, * operators (bonus for *)

These operators should perform exactly as their counter parts in int, i.e.

```
unsigned long long a = 1000000;
UnsignedLargeInt i1(a);
UnsignedLargeInt i2(100);
UnsignedLargeInt i3;
i3 = i1 + i2;
cout << i3; // should output 1,000,100;
```

Each of these operators should take two "const UnsignedLargeInt&" as arguments and return a UnsignedLargeInt. These operators should be designed and implemented as friend operators. It has no requirement to overload these operators. For example:

```
unsigned long long a = 1000000;
UnsignedLargeInt i1(a);
UnsignedLargeInt i2;
i2 = i1 + 100; //compiler will transfer 100 to a UnsignedLargeInt,
               //so we do not have a + operator to take int as argument.
cout << i2; // should output 1,000,100;
```

2.3 +=, *= operators (bonus for *=)

These operators should perform exactly as their counter parts in int, i.e.

```
unsigned long long a = 1000000;
UnsignedLargeInt i1(a);
UnsignedLargeInt i2(100);
i1 += i2;
cout << i1; // should output 1,000,100;
```

Each of these operators should take a "const UnsignedLargeInt&" as argument and return "UnsignedLargeInt&". It has no requirement to overload these operators.

2.4 boolean operators !, !=, ==, <, >, <=, >=

These operators should perform exactly as their counter parts in int, i.e.

```
unsigned long long a = 1000000;
UnsignedLargeInt i1(a);
UnsignedLargeInt i2(100);
if (i2 < i1) cout << "i2 is smaller" << endl;
else cout << "i1 is smaller" << endl;
```

Each of these operator should take two "const UnsignedLargeInt&" as arguments (except the unary operator ! which will take no argument) and return "bool". These operators (except !) should be designed and implemented as friend operators. It has no requirement to overload these operators.

2.5 incremental operators ++

These operators should perform exactly as their counter parts in int, and should have two versions (pre-increment and post-increment), i.e.

```
unsigned long long a = 1000000;
UnsignedLargeInt i1(a);
i1++;
cout << i1; //should output 1,000,001
cout << ++i1; //should output 1,000,002
```

2.6 << operator

Output the UnsignedLargeInt in a nice format, i.e.

```
unsigned long long a = 10000000000000000;
UnsignedLargeInt i1(a);
cout << i1;
//The output should look like:
10,000,000,000,000,000
```

This operator should be friend (just like what we did in the previous project).

2.7 Other requirements

No need to overload the ">>" operator.

Comment you code! Indent your code!

3 Files to turn in:

You need to turn in four files: makefile, main.C, UnsignedLargeInt.C, UnsignedLargeInt.h. This time, I will not provide a UnsignedLargeInt.h, you need to design your own.

You should have a main.C that can test all your operators.

4 Grading

The total points is 14, you will get full credit if all constructors, destructor and operators work correctly and there is no major bug in your code. You will get 6 bonus points for * and *= operators.

The grade will be based on the implementation of individual operators. However, 2 points are reserved for the implementation of the main function.

No late turn in is acceptable, any late turn in will be given 0 point.

Your code must be compilable on Linux/Unix, any code that cannot be compiled by g++ will automatically get ZERO point. This suggests that you should implement and test the operators one by one, and do not turn in the operators that can not be compiled.