COP3330 Programming Assignment 3

Deadline: Sunday Mar 3rd, 2019

Objective: Upon completion of this program, you should have more experience with **friend functions** and **overloading basic operators** in C++ classes.

Overview: A well-known drawback of the native 32-bit `int` type in C++ is that it cannot store large integers of more than 10 decimal digits. In this project, you will implement a **bigint** class that behaves like the **unsigned int** type, but can handle unsigned integers of up to **45 digits**. Big integers have found widely varying applications in scientific and societal domains, such as recording distances between stars or planets in the universe, and keeping track of the number of atoms or cells in organisms.

Details:

You are to build a class named ***bigint*** that can deal with up to 45 decimal digits of unsigned integers. The *bigint* class is specified as follows.

1. The *bigint* class should have a five-element array, (*int v[5];*), as its private data member. Each element can store 9 decimal digits in the range of 0 up to 999999999. For example, to store the integer 1,300,000,000,000,900,000,000,100,000,000. You should have the internal storage as v[0] = 100000000 (the last nine digits), v[1] = 900000000, v[2] =0, v[3]= 1300, and v[4] = 0. This way, we can store successfully a large enough integer with up to 45 digits.

2. The bigint class should support the following **constructors**:

```
bigint();          // default constructor, set the value to be 0

bigint(int x0);         // set the value to be x0

bigint(int x0, int x1); // set the value to be x1*10^9+x0

// set the value to be x2*10^18+x1*10^9+x0
bigint(int x0, int x1, int x2);

// set the value to be x3*10^27 + x2*10^18+x1*10^9+x0
bigint(int x0, int x1, int x2, int x3);

// set the value to be x4*10^36 + x3*10^27 + x2*10^18+x1*10^9+x0
bigint(int x0, int x1, int x2, int x3, int x4);
```

3. To make bigint behave like int, you must **overload the following operators**:

a. **The extraction operator >>** for reading a bigint number from an input stream. The input number can have up to 45 digits. Following are two bigint numbers that can appear as input:

```
1300000000000900000000100000000
999999999999999999999999999999999999999999999
```

To implement the >> operator, you can first read the stream of digits into a character array (or string) and then manually convert it into a bigint object. Here you may use cin.get() or getline() functions to accept the input from input streams.

b. **The insertion operation <<** for output a bigint number to an output stream. The number must be output like a regular int number starting from most significant bits(v[4]) all the way down to least significant bits (v[0]). Here you may need to output potentially 5 integers. You may use setw(9) and setfill('0') to output each value besides the most significant non-zero integer. If so, you have to have '#include <iomanip>' in your code in order to use these functions. The following line is an example to use these functions.

```
s << setfile('0') << setw(9) << a.v[j];
```

For instance,

```
bigint x(1, 123);

cout << x << endl;
```

The expected output will be 123000000001 (the last 1 is padded with eight 0s on its left).

c. **Two arithmetic operators** + and – ;
For addition (a+b), please pay special attention to **the carry bit** to the higher significant bits. Moreover, adding two bigints, a and b, may result in overflows, if a.v[4]+b.v[4] > 999999999. In this case, we will omit the carry bit (to the even higher significant bits), and only keep the remaining raw values as is. For example:

```
  900000000 000000000 000000000 000000000 900000000
+ 900000000 000000000 000000000 000000000 900000000
= 800000000 000000000 000000000 000000001 800000000
```

Note here 900000000 + 900000000 = 1800000000, while the leading carry bit is omitted because of overflow.

For subtraction (a-b), if a < b, the result will be 0, because we only use bigint to maintain **unsigned** integers. If a >= b, we follow the regular arithmetic logic of subtraction for calculation, and please pay special attention to **the carry bit** during the computation.

d. **Six comparison operators** <, >, <=, >=, ==, and !=
bigint objects are still integers, so they can be compared and ordered. Overloading these operators will support comparisons between different bigint objects.

Testing:

Once the class is implemented, you can test your implementation using the sample `proj3.cpp` program. You should also write other test programs to do a thorough test of your bigint implementation. If user input is required (for >> operator), you can redirect `proj3_test.txt` as the standard input (otherwise, you have to input manually the overly long numbers each time, which are extremely tedious):

```
./proj3 < proj3_test.txt
```

General Requirement:

1. You need to implement the bigint class all by yourself! You may use the `istream`, `ostream`, `iostream`, `iomanip`, `string` libraries in your implementation. DO NOT use any other C++ standard libraries, or any third-party libraries or API calls;
2. No global variables, other than constants!
3. All member data of your class must be private;
4. The const qualifier should be used on any member functions and parameters where it is appropriate;
5. You only need to do error-checking that is specified in the descriptions above. If something is not specified (e.g. user entering a letter where a number is expected), you may assume that part of the input will not be evaluated;
6. User input and/or screen output should only be done where described (i.e. do not add in extraneous input/output);
7. When you write source code, it should be readable and well-documented.

Submission:

Program submissions should be done through Canvas. Do not send program submissions through e-mail -- e-mail attachments will not be accepted as valid submissions!

For Assignment 3, submit a compressed package (named **proj3.tar.gz**) including the following files

bigint.h
bigint.cpp
makefile (the final executable is named **proj3**, and the file containing main() function is named **proj3.cpp**)
readme (optional)

Make sure your filenames are these exact names, and do not submit your proj3.cpp file (or any other main program you might create for testing purposes).