3N+1 BigInt - Prog 7 (20 points)
CECS 325-01 – System Programming with C++
Spring 2023
Due: May 3, 2023

YouTube video: https://youtu.be/094y1Z2wpJg

The **Collatz conjecture** is one of the most famous unsolved problems in mathematics. The conjecture asks whether repeating two simple arithmetic operations will eventually transform every positive integer into 1. It concerns sequences of integers in which each term is obtained from the previous term as follows:

If the previous term is even,
the next term is one half of the previous term.
If the previous term is odd,
the next term is 3 times the previous term plus 1.

The conjecture is that these sequences always reach 1, no matter which positive integer is chosen to start the sequence.

It is named after mathematician Lothar Collatz, who introduced the idea in 1937.

It is also known as the **3*n* + 1 problem**.

As of October 2018 the largest number verified for Collatz conjuncture is $2^{100000} - 1$.

Your task is to write a program that we can use to explore the 3n+1 problem.

In our Program 6 we expected that we might need to check for overflow. In this program we are going to create a BigInt class so there is no possibility of overflow. I'm including a test driver below that you can use – but notice these numbers can get big. So YOU need to create a BigInt class that supports unlimited size numbers. The testdriver below suggests what functions the BigInt class needs. Here is the BigInt header file you can use to get started:

```
class BigInt
{
      private:
            vector<char> v;               // notice this is a vector of char… not int!!!
      public:
            BigInt();                     // default constructor - set value to 0
            BigInt(int);                  // int constructor
            BigInt(string);               // string constructor
            BigInt operator+(BigInt);     // add 2 BigInts, return the sum
            BigInt operator++( );         // prefix increment
            BigInt operator++(int);       // postfix increment
            BigInt operator*(BigInt);     // multiply operator
            // BigInt operator /(BigInt); // divide operator - optional.
                                          // Not needed if half( ) is used.
            BigInt half();                // return half the value
            bool isOdd();                 // true if the number is odd
            bool isEven();                // true if the number is even
            bool operator==(BigInt);      // true if 2 BigInts are equal
      friend ostream& operator<<(ostream&, const BigInt&); // cout << BigInt
};
```

In this assignment you will implement operator overloading in the BigInt class as shown above. Below is the testDriver you are required to use.

```
// 3n+1 test driver

#include <iostream>
#include <limits.h>
//#include "BigInt.h"  // please include the BigInt class in the main file
using namespace std;

// create struct to store all details of 3n+1 sequences
struct ThreeNp1 {
    BigInt start;
    BigInt steps;
    BigInt max;
    BigInt odd;
    BigInt even;
};

// utility function to see details of 3n+1 sequence
// notice that all values are BigInt… cout << BigInt
void print(ThreeNp1 temp)
{
    cout << "start:"<<temp.start<<endl;
    cout << "steps:"<<temp.steps<<endl;
    cout << "max:"<<temp.max<<endl;
    cout << "odds:"<<temp.odd<<endl;
    cout << "evens:"<<temp.even<<endl;
}

// recursive function to find all details about 3n+1 sequence
// Function has a default parameter as the 3rd parameter
void findThreeNp1(BigInt n, ThreeNp1 & Np1, bool printSteps = false)
{
    if (printSteps)
    {
        cout << "->"<<'('<< n <<')';
    }
    if (Np1.max < n)                 // BigInt::operator<( )
        Np1.max = n;                 // No need to overload - C++ provides operator=( )

    if (n == BigInt(1))              // BigInt::operator==( )
    {
        return;                      // we are done
    }
    else if (n.isEven())             // BigInt::isEven()
    {
        Np1.even++;                  // BigInt::operator++( )
        Np1.steps++;
        //findThreeNp1(n/2,N, printSteps);            // BigInt::operator/( ) - Hard…
        findThreeNp1(n.half(), Np1, printSteps);      //BigInt::half( ) - Easy
    }
    else if (n.isOdd())              // BigInt::isOdd( )
    {
        Np1.odd++;
        Np1.steps++;
        BigInt tempN(n);             // BigInt constructor
        findThreeNp1(tempN*BigInt(3)+BigInt(1), N, printSteps); //BigInt::operator*( )
                                                                //BigInt::operator+( )
    }
    else
    {
        cout << "How the hell did I get here?\n";
        return;
    }
}

//https://en.wikipedia.org/wiki/Collatz_conjecture
int main()
{
    BigInt MAX(INT_MAX);
    cout << "The largest integer is "<< MAX<<endl;
    cout << "Twice the largest integer is "<< MAX + MAX << endl;
    BigInt start(INT_MAX);          // BigInt constructor - use for submission
    //BigInt start(12);             // BigInt constructor – use for testing
    bool printSteps = true;
    ThreeNp1 N = {start,0,0,0,0}; // initialize N
    findThreeNp1(start, N, printSteps); // print out the steps
    cout << endl;
    print(N);

    return 0;
}
```

Expected output for start = 12 (this is for testing – do not submit):

```
The largest integer is 2.1474836e9
Twice the largest integer is 4.2949672e9
->(12)->(6)->(3)->(10)->(5)->(16)->(8)->(4)->(2)->(1)

start:12
steps:9
max:16
odds:2
evens:7
```

Expected output for start = INT_MAX and printSteps = false (Submit this):

```
The largest integer is 2.1474836e9
Twice the largest integer is 4.2949672e9

start:2.1474836e9
steps:450
max:1.2353467e15
odds:162
evens:288
```

Notice that when using "cout <<" to print a BigInt, the number will print all digits if there are 8 or less digits in the number. However, if there are more than 8 digits, the number will print in exponential notation with 8 significant digits.
Exampe:
BigInt x(12345678);
BigInt y(123456789);
BigInt z("123456789123456789");
cout << x;   // prints 12345678
cout << y;   // prints 1.2345678e9
cout << z;   // prints 1.2345678e17

Include the BigInt class inside the testDriver.cpp – in other words DO NOT have separate .h and .cpp files for the BigInt class.

What to submit:

1) The testDriver.cpp
2) Screenshot of the Demo Program output.