# Assignment 3: Big Integer

Written by Ashok after looking at many assignments that actual professional teachers use.

**This assignment will be due Friday August 3rd** (i.e. you get a few more days to complete this assignment).

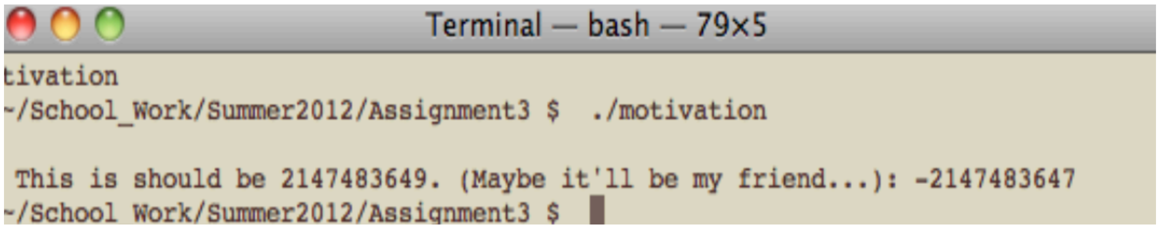
I picked an easier version of the assignment (partly because you have a midterm this week) BUT START EARLY! This will be your hardest assignment!

The Psuedocode is due Thursday at Midnight (but if you're a little late this week it's all good because of the midterm)

So I was originally planning on not giving you kiddos an assignment this week; I was super excited. No grading. No emails. No "Did you include iostream? Are you sure? You might want to check if you included iostream. Naw just double check, I'll wait. . . Oh you did? Really?! Crap." So suddenly I had this free weekend, and as I do with every free weekend, I busted out C++ and started running code (social life's not going so hot…) and I tried the following:

```
****
#some crazy includes
#using stds (but like, the good kind)

int main()
{
        int i=2147483647;

        i+=2;

        cout<<"This is should be 2147483649. (Maybe it'll be my
friend...): "<<i<<endl;

        return i;
}

****
```

And then I compiled it, no errors, and ran it and got the following in my terminal:



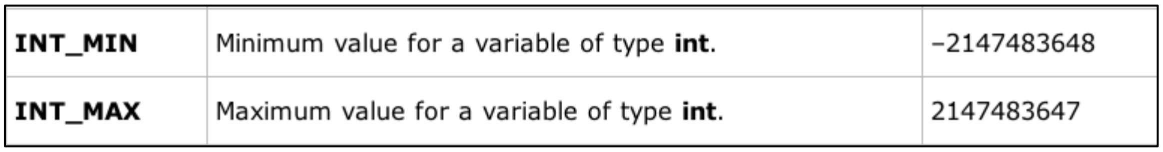
```
tivation
~/School_Work/Summer2012/Assignment3 $  ./motivation

 This is should be 2147483649. (Maybe it'll be my friend...): -2147483647
~/School_Work/Summer2012/Assignment3 $
```

Figure 1: Finding Friendship?

So I was expecting 2147483649, and I got -2147483647. Whatevs ya'll, computers can't be perfect. Ain't. no. thang. Frealz, I'm only off by 4294967296. Pshht drop in the bucket for laid back people like us right?!

But then I got to thinking, even though this error is super trivial and so tiny no one would ever notice it, what the H is going on?!

Well, a cursory google search gives us this – the minimum and maximum values for a signed int (i.e. signed meaning the int can be positive or negative):



| | | |
|---|---|---|
| **INT_MIN** | Minimum value for a variable of type **int**. | –2147483648 |
| **INT_MAX** | Maximum value for a variable of type **int**. | 2147483647 |

Figure 2: The Limits Of Friendship?

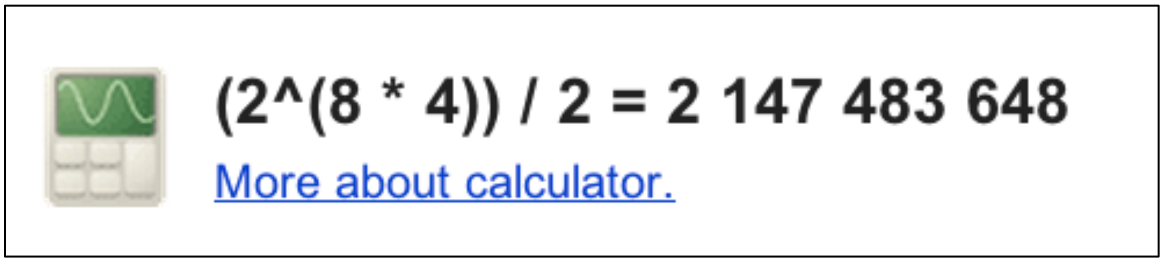
Looking past the astonishing coincidence that the number I tried to add to just so happened to be the upper limit of signed ints in C++, Why did we get such a weird solution in the terminal above?!

The answer lies in the term "integer overflow."
From Wikipedia:
***

*If the variable has a signed integer type, an overflow can cause its value to wrap and become negative. This overflow violates the program's assumption and may lead to unintended behavior. Similarly, subtracting from a small unsigned value* *may cause it to wrap to a large positive value which may also be an unexpected behavior. Multiplying or adding two integers may result in a value that is non-negative, but unexpectedly small. If this number is used as the number of bytes to allocate for a buffer, the buffer will be allocated unexpectedly small, leading to a potential buffer overflow.*****

So for signed ints, if we go over that max int value in Figure 2, we wrap around to the negative side. Technically speaking, for signed ints in C++ the behavior is undetermined which means it usually wraps around but it's not guaranteed to. If you didn't understand that don't worry, just know that our ints have a hard cap.

A total aside: An int is 4 bytes, each byte has 8 bits (8*4=32), so we can actually calculate this hard cap number. We can figure the total number we can represent by taking 2 to the 32 (because each bit can be 1 or 0, we have 32 bits) and then dividing by 2 (i.e. dividing this guy "equally" among the negative and positive numbers).



(2^(8 * 4)) / 2 = 2 147 483 648

More about calculator.

Figure 3: Limits on Friendship 2?

So in this assignment we are going to make a new type (i.e. a new class) called **"bigInt"** that allows us to theoretically have a number that is 2147483648 DIGITS LONG (for comparison, that number in Figure 3 is only 9 digits long).

There are many ways we can implement this. Our basic strategy will be to store each digit as a dynamically allocated cstring array that resizes to the amount of memory we need (you need to include <cstring>, you can read about cstring on pages 197 tells you what they are (i.e. how to make them, they're really easy)

DEFINITELY READ THIS, 794 tells you premade c++ functions that you can use with this DEFINITELY LOOK AT THIS because it'll make your life easier. A complete list of functions that you can use with your cstring are
http://www.cplusplus.com/reference/clibrary/cstring/

This lab gives a nice intro to them too:
http://1300.dupland.com/recitation-6.
(more on cstrings later)

Included in this are 3 files, A header file of all the functions you should implement, a .cpp file that includes the skeleton of all these functions (with a few of them implemented), and a main.cxx file that will allow you to test your program.

Here is what the .h file looks like:

```
***
#ifndef BIGINT_H

#define BIGINT_H

#include <iostream>

class bigInt

{
        public:
        //constructors
        //default sets it to 0 with an array of size 1
        bigInt(); //
        //passes in a char array to initialize value
        bigInt(const char initialValue []); //
        //passes in an integer
        bigInt (int initialValue); //
        //passes in a double
        bigInt (double initialValue); //
        //copy constructor
        bigInt (const bigInt& toCopyFrom);//

        //Getter
        //returns a pointer to a COPY of our array of char*'s
        char* getBigIntArray() const;//

        //operators += -=
        void operator += (const bigInt& intToAdd); //
        void operator -= (const bigInt& intToSubtract);
        void operator =(const bigInt& toEqual);//

        //nonmember friends istream and ostream
        friend std::istream& operator >>(std::istream& ins, bigInt&
target);//

    friend std::ostream& operator <<(std::ostream& outs, const
bigInt& source);//

        private:
        //numDigits-- returns current number of digits

        //private members
        char* bigIntArray; //where all our numbers reside
};
//NONMEMBERS
//operator+
bigInt operator +(const bigInt &firstInt, const bigInt &secondInt);
//
//operator-
bigInt operator -(const bigInt &firstInt, const bigInt &secondInt);
//operator==
bool operator ==(const bigInt &firstInt, const bigInt &secondInt);
//operator !=
bool operator !=(const bigInt &firstInt, const bigInt &secondInt);
//operator <=
bool operator <=(const bigInt &firstInt, const bigInt &secondInt);
//operator >=
bool operator >=(const bigInt &firstInt, const bigInt &secondInt);
```

```
//operator <
bool operator <(const bigInt &firstInt, const bigInt &secondInt);
//operator >
bool operator >(const bigInt &firstInt, const bigInt &secondInt);

#endif
```

***

We will assume all of our **bigInts are non-negative**. For the subtraction we will assume that the first number is always bigger than the second number.

Three functions have been provided for you, the cin operator (which cheats a little bit to be implemented but will work for your purposes), the cout operator (which you've implemented enough of this semester), and the getBigIntArray function which returns a pointer to a dynamically allocated copy of a given bigInt's bigIntArray (think about it). So, for example, in the copy constructor you can use this member function to set the bigIntArray of the "object you're copying-into" to the bigIntArray of the "object you're copying from" (and it will be it's own copy).

Feel free to add any helper functions you want to this class but **make sure your program works with the main.cxx provided.**

Here are some functions I found helpful:
Cstring:
-**strlen**(pass in some cstring): returns the length of the cstring NOT including the null character

-**strcpy**(target,source): copys one cstring into another including the null character. Make sure target has the correct allocated amount before doing this.

-**changing a character to an int**:
you can do this digit by digit. Take a digit of your intArray and minus the '0' ascii character like so

```
(temp1[firstIndex]-'0')  //where temp1 is a char* that has a
                         //number in it and ‘0’ is a zero in single
                         //quotes
```

**-changing an int to character**
**Apparently this works to**

```
intToConvert=5;
char exampleChar= ‘0’+intToConvert;  //the ‘0’ is a zero again
```

there are other methods: Some people use the “itoa” function for this or use a stringstream for this. Feel free to google them an use them if you’d like.

you can also check this out:
http://stackoverflow.com/questions/4629050/convert-an-int-to-ascii-character

So right now you’re probably thinking, this is great and all but how do I attack this monstrosity? In the .cpp file I’ve put the implementation functions in order of which the main file tests them. The main file starts all commented out except for the default constructor test. As you implement the .cpp functions, you can uncomment that particular test in main.cxx. In this way you can methodically break this assignment down (sometimes I ask you to write your own test).

Also I’ve put hints in the comments of the .cpp file to make your life easier. You can use these comments as a guide or devise your own solution. If you do something (or attempt something) that you think is novel, please tell us in the write up so we can give you extra credit etc.

If you finish and want an additional challenge, you can try to implement multiplication and division for extra credit too (there is pseudocode out on the internet that can help get you started). Finally, please tell me if there’s anything wrong with any of these files or this writeup ASAP and I’ll do my best to correct them/

inform the class. Start as soon as possible-- it’s very easy to make mistakes on such a big project.