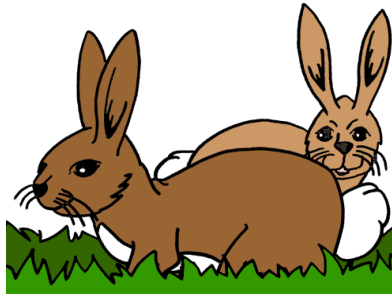


# CSCI 135 Software Design and Analysis I

## Project A

### More rabbits than you can imagine

Saad Mneimneh  
Visiting Professor  
Hunter College of CUNY



The goal of this project is to compute the Fibonacci for arbitrarily large numbers. The Fibonacci of a number  $n$  is defined as:

$$Fib(n) = \begin{cases} 1 & n \leq 1 \\ Fib(n-1) + Fib(n-2) & n > 2 \end{cases}$$

We have seen in class how to implement the Fibonacci function efficiently using both recursion and iteration. However, while efficiency is a concern, another concern here is that Fibonacci grows really fast. Consider the following for instance:

```
int fib(int n) {...}
```

Soon, `int` will not be enough bits to represent the Fibonacci of a large number. With `int` being the return type, Fibonacci of 44 is the largest we can compute, which is 1836311903. The goal of this project is, therefore, to implement the Fibonacci function in the following way:

```
LargeInt fib(int n) {...}
```

`LargeInt` is a user defined class to represent large integers. Here's a possible outline for such a class:

```

class LargeInt {
    int * digits; //array of digits
    int len; //number of digits

public:
    LargeInt();
    LargeInt(int n);
    LargeInt(int n, int len);
    LargeInt(const LargeInt& n);
    ~LargeInt();

    LargeInt& operator=(const LargeInt& n);
    int length()const {return len;}
    void expand(int len);
};

const LargeInt operator+(const LargeInt& m, const LargeInt& n);
ostream& operator<<(ostream& s, const LargeInt& n);

```

The default constructor creates a `LargeInt` with the value zero (hence only one digit is enough). The second constructor creates a `LargeInt` with the value  $n$ ; therefore,  $\lfloor \log_{10} n \rfloor + 1$  digits are required. The third constructor creates a `LargeInt` with the value  $n$  but makes sure that the representation contains at least  $len$  digits. The member function `expand` adds enough 0 digits to the left to make the number of digits equal to  $len$  without changing the value of the `LargeInt`. The rest of the functions should be self explanatory.

The bulk of the work is `operator+`. One possible implementation of this operator is the following: Let  $m$  and  $n$  be the two `LargeInt`s to be added. Create two `LargeInt`s  $a$  and  $b$  equal to  $m$  and  $n$  respectively, but with  $len = \max(m.length(), n.length())$  digits each. Let  $c$  be the result of the addition and construct  $c$  initially using the default constructor. Then `expand`  $c$  to  $len$  digits, and update the digits of  $c$  to correspond to those of  $a + b$ . If there is an outermost carry, `expand`  $c$  by one more digit and set that digit accordingly.

Your program should illustrate the correct operation of this class. In particular, you must output the Fibonacci of some very large numbers.

**Note:** An alternative implementation of class `LargeInt` can use a C++ vector instead of an array. In this case, one would not worry about declaring a destructor, a copy constructor, or an assignment operator.