**CSULB - CECS 282-05 (C++ for Java Programmers) Spring 2021**

**Program #5 – goldRabbits: BigInt using vectors and maps**

**Due: April 20, 2021**

I got a pair of rabbits for my birthday. I did some research and found out that apparently rabbits multiply like… well, Rabbits! So I have a great idea that will help me retire. I’m going to start a rabbit ranch. I need to plan for the future and determine how much land and rabbit food I will need for my rabbits.

Here are the Rabbit Rules:

1. New-born rabbits take one month to mature.
2. One pair of mature rabbits will have a pair of baby rabbits after one month.
3. Mature rabbits will continue to produce a new pair of baby rabbits every month.
4. Rabbits never die

I plan to retire in 10 years. How many rabbits will I have by then?

I did some hand calculations and I came up with this:

At the start of month 0, I receive the baby rabbits so I have **1 pair** of rabbits.

At the start of month 1, the baby rabbits are grown and the female becomes pregnant. But I still have only **1 pair** of rabbits.

At the start of month 2, the first pair give birth to 1 pair of rabbits, so now I have **2 pairs** of rabbits. The first pair (the adults) immediately become pregnant again.

At the start of month 3, the second pair is now mature and the first pair give birth again to 1 pair of rabbits. This gives me **3 pairs** of rabbits.

At the start of month 4, the first pair gives birth to a new pair, the second pair gives birth to a new pair, and the third pair are all grown up and ready to start family life. So now I have **5 pairs** of rabbits.

By now I was getting confused with counting rabbits, but I noticed a pattern. It seemed like each new month the total number of pairs of rabbits was equal to the total number from the month before plus the total number from 2 months before. That’s really cool that I can predict the number of rabbit pairs with an equation.

I’m gonna call it “Master Gold’s Really Cool Formula for Rapid Rabbit Population Growth”, or just GoldRabbits for short. (Apparently some Italian dude from the year 1202 named Fibonacci came up with something similar – but I think mine is better…)

Here’s the definition of GoldRabbits:

// this function predicts the ***number of rabbit pairs*** I will have after n months

**int GoldRabbits( int n )**

**{**

**if (n == 0 || n == 1)**

**return 1;**

**else**

**return GoldRabbits( n-1 ) + GoldRabbits( n–2 );**

**}**

I wanted to see how many rabbits I would have in 10 years so I wrote this little program:

#include <iostream>

#include <time.h>

#include <iomanip>

using namespace std;

int goldRabbits(int); // prototype or signature

int main()

{

int const months = 12 \* 10; // this is 10 years or 120 months

int start = time(0); // number of seconds since Jan 1, 1970

for(int i=0; i<months; i++)

{

int current = time(0); // number of seconds since program started

cout << setw(5)<<current-start<<":"; // print elapsed seconds

cout << " GoldRabbits("<<setw(2)<<i<<") = ";

cout << setw(11)<< **goldRabbits(i)** <<endl;// the call to goldRabbits

}

}

**// this is the goldRabbits function**

int goldRabbits(int n)

{

if (n==0 || n==1)

return 1;

else

return goldRabbits(n-1) + goldRabbits(n-2);

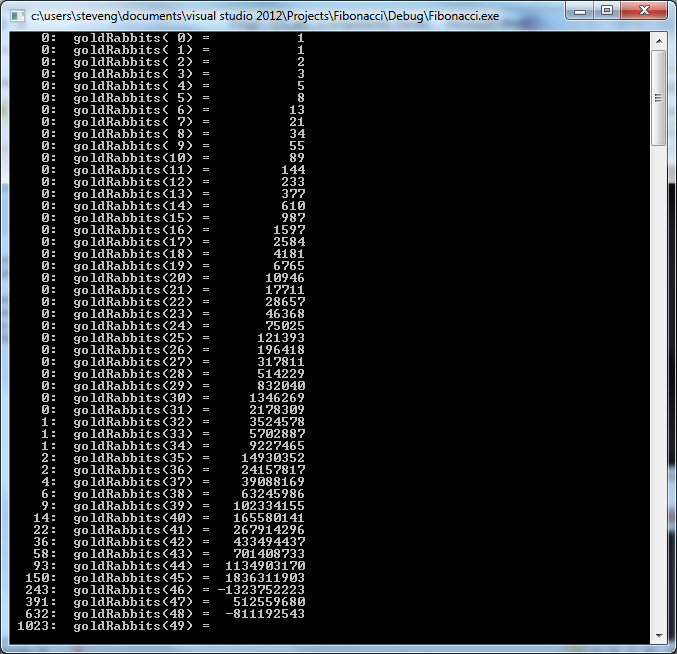
}

After 17 minutes, (or 1023 seconds) here is my output:

The first column is the number of seconds that has elapsed since the program started.

The second column shows the month number (how many months into the rabbit farm).

The third column shows the numbers of rabbits I currently have on the rabbit farm.



As you can see, I have a couple problems:

1. For some reason, the number got real big – it was approaching 2,147,483,647 at goldRabbits(45) – (Actual value was 1,836,311,903 when it suddenly went negative at goldRabbits(46)… **That’s weird. Am I right? What is going on????**
2. It’s taking a long, long time to calculate. At this rate, I will be retired before the calculation is complete…

Your assignment is to solve these 3 problems for me.

1. Write the goldRabbits(int n) function that throws an exception if it detects integer overflow. The exception will throw a string that indicates the input number (n) which caused the overflow.
2. You need to create a new Class called BigInt that will allow me to calculate the goldRabbits(120), or even goldRabbits(2000) so that my grand daughter can take over the rabbit farm. This is the prototype:
   1. **BigInt goldRabbits(BigInt )**
   2. **Use the STL vector to store your digits so you can have BigInts that are infinitely long.**
   3. **Use the char data type to store “digits” in the vector**

1. You need to modify the function goldRabbits so that it can calculate goldRabbits(2000) before we all die.
   1. **Use the STL map to store the value of goldRabbits(n) so you can quickly use that value when you are trying to calculate goldRabbits(n+1). You will also need to use “static”. In other words, when you try to calculate goldRabbits(n), you should already have goldRabbits(n-1) and goldRabbits(n-2) stored and saved in your static map.**
   2. You need to create a new class called BigInt so you can handle numbers greater that 2^31-1 (which is the largest positive integer that C++ can handle using the native data type int.) BigInt needs to handle addition and subtraction, constructor, copy constructor, assignment operator, destructor, and other functions as needed. But don’t create a function if you don’t need to. You will use the **vector** to store the digits or characters of your BigInt. **Your storage for the BigInt class will be a vector of chars**. You will treat the chars as if they were ints. The largest int value you can store in a char is 255 – which is way big enough since the largest digit is only 9. You need to implement the minus operator, but you can assume that your result will never be negative. **In other words you do not need to support a negative BigInt**.
2. **You will write only one file for this program**. The header portion for the BigInt class will be declared above main. The implementation of the BigInt class will be written after main. Any additional functions you use will follow the same structure – the prototypes will be before main and the implementation will be after main. Here is an example of how you would write a program that handled Cups:

**#include <iostream>**

**using namespace std;**

**class Cup{ // this is the header portion of the Cup class**

**private:**

**int value;**

**public:**

**Cup(int);**

**friend ostream & operator<<(ostream &, const Cup &);**

**Cup operator+(Cup);**

**};**

**int main() // this is the main function**

**{**

**Cup c1(23);**

**cout << c1<< endl; // prints 23**

**Cup c2(c1); // copy constructor**

**Cup c3 = c1 + c2;**

**cout << c3; // print 46**

**}**

**// implementation of Cup member and non-member functions below**

**Cup::Cup(int x){**

**value = x;**

**}**

**ostream & operator<<(ostream &out , const Cup & C){**

**cout << C.value;**

**}**

**Cup Cup::operator+(Cup C){**

**Cup temp(\*this);**

**temp.value += C.value;**

**return temp;**

**}**

On demo day, you will run the following program. Notice that this program uses the slow version of goldRabbits and also calculates using int. You have to rewrite the goldRabbits( BigInt) function so it runs fast and uses BigInt. Also notice there are 2 ways to display a BigInt.

BigInt B1(“123456789123456789123456789”);

cout << B1; // this number is more than 12 digits so it will display in scientific notation

B1.print(); // this will display all digits in the BigInt

#include <iostream>

#include "BigInt.h"

#include <map> // you will need this for the goldRabbits function

using namespace std;

int goldRabbits(int);

BigInt goldRabbits(BigInt);

void pause() {cout << "Press Enter to continue..."; getchar();}

int main()

{

BigInt B1("123456789123456789123456789123456789");

BigInt B2(B1);

BigInt MAX(INT\_MAX);

cout << "B1:"<<B1<<"\nB2:"<<B2<<"\nMAX:"<<MAX<<endl;

pause();

cout << "\nB1:";

B1.print();

cout << endl;

pause();

// for(BigInt i=0; i<=3000; i++) // uncomment this

for(int i=0; i<=3000; i++) // comment this out

{

cout << "\ngoldRabbits("<< i <<") = " << goldRabbits(i);

}

pause();

cout<< "\nThis is the value of goldRabbits(3000)\n\n";

BigInt gR3000 = goldRabbits(BigInt(3000));

gR3000.print();

pause();

int n = 500;

try{

cout << “The value of goldRabbits(n) is:”<<goldRabbits(n)<<endl;

}

catch(string error)

{

cout << error << endl;

cout << “Transitioning to BigInt\n”;

cout << goldRabbits(BigInt(n));

}

pause();

}

// Modify this function so that it accepts BigInt as input parameter and returns a BigInt

// and uses a map for speed

int goldRabbits(int n)

{

if (n==0 || n==1) // base case

return 1;

else

return goldRabbits(n-1) + goldRabbits(n-2); // general case

}

**This is the desired output:**

B1:1.2345678912e36

B2:1.2345678912e36

MAX:2147483647

Press Enter to continue...

B1:123456789123456789123456789123456789

Press Enter to continue...

goldRabbits(0) = 1

goldRabbits(1) = 1

goldRabbits(2) = 2

goldRabbits(3) = 3

goldRabbits(4) = 5

goldRabbits(5) = 8

goldRabbits(6) = 13

goldRabbits(7) = 21

(some lines removed)

goldRabbits(56) = 365435296162

goldRabbits(57) = 591286729879

goldRabbits(58) = 956722026041

goldRabbits(59) = 1.5480087559e13

goldRabbits(60) = 2.5047307819e13

goldRabbits(61) = 4.0527395378e13

goldRabbits(62) = 6.5574703198e13

goldRabbits(63) = 1.0610209857e14

goldRabbits(64) = 1.7167680177e14

goldRabbits(65) = 2.7777890035e14

goldRabbits(66) = 4.4945570212e14

(many lines removed here)

goldRabbits(2994) = 3.702521137e626

goldRabbits(2995) = 5.990805043e626

goldRabbits(2996) = 9.693326181e626

goldRabbits(2997) = 1.568413122e627

goldRabbits(2998) = 2.537745740e627

goldRabbits(2999) = 4.106158863e627

goldRabbits(3000) = 6.643904603e627

Press Enter to continue...

This is the value of goldRabbits(3000)

664390460366960072280217847866028384244163512452783259405579765542621214161219257396449810982999820391132226802809465132446349331994409434926019045342723749188530316994678473551320635101099619382973181622585687336939784373527897555489486841726131733814340129175622450421605101025897173235990662770203756438786517530547101123748849140252686120104032647025145598956675902135010566909783124959436469825558314289701354227151784602865710780624675107056569822820542846660321813838896275819753281371491809004412219124856375121694811728724213667814577326618521478357661859018967313354840178403197559969056510791709859144173304364898001

Press Enter to continue...

The value of goldRabbits(500) is: Overflow Error – goldRabbits overflowed using 47

Press Enter to continue...

What to submit on demo day:

1. One single file which includes all the source code for this project.
2. One single screenshot similar to the above. In order to fit the output on a single screen, modify the for loop so that only the values of goldRabbits(x) print out where x is in the set of lines that are shown above.