Table of Contents » Part 1: foundations » Programming assignment 1: Linked-list integers » Update: requirements reduced

Update: requirements reduced

We will do a bit less

. . . because time is tight. You do LLInt, but now make it unsigned. I'll give you the Fibonacci magic.

- Restriction 1: zero or positive. LLInt only has to accept nonnegative inputs, and represent nonnegative values.
- Restriction 2: no subtraction. The LLInt class should support **addition** and **multiplication**, but is not required to do subtraction anymore.
- ~~Restriction 3 (possible): Fibonacci number 10,000 might be out of reach for us, and I might reduce that.~~ (Not necessary)
- I'll give you some code to do the Fibonacci computation (link below). We will discuss it in class.

```
/*
 * filename     Fibonacci.java
 * language     Java 1.7.0_05
 * author       Andrew Predoehl
 * license      One copy to each student of CS 345 Summer 2016,
 *              but no further copying.
 * date         16 May 2016
 * description    Compute arbitrarily large Fibonacci number n,
 *              given arbitrary-precision nonnegative integer LLInt.
 *
 * Written as part of the summer 2016 session of CSc 345,
 * at the U. of Arizona.
 *
 * Students:  you might need to make a few tweaks -- "throws" statements, or
 * try-catch blocks, or changes based on your add() and multiply() operations.
 * All that is fine, just submit your customized version with your project.
 */



/**
 * Create and compute with 2x2 matrices of BigIntegers.
 * This class supports a very simple constructor and just a few methods, but
 * it does have a fast method to raise a matrix to an arbitrary nonnegative
 * integer power.
 */
class Mat2x2 implements Cloneable {

    /* matrix entries named by their row and column (1-based like we do in
     * math)
     */
    private LLInt a11, a12, a21, a22;

    public LLInt get11()
    {
        return a11;
    }
    public LLInt get12()
    {
        return a12;
    }
    public LLInt get21()
    {
        return a21;
    }
    public LLInt get22()
    {
        return a22;
    }

    /** Basic ctor for your matrix.  Reads in the matrix entries in row
     * major order.
     */
    public Mat2x2( LLInt k11, LLInt k12, LLInt k21, LLInt k22 )
    {
        a11=k11;
        a12=k12;
        a21=k21;
        a22=k22;
    }

    /** Compute and return the matrix product of the given matrix with
     *     this matrix.
     *  @param m    The matrix to be multiplied on the RIGHT side of this.
     *  @return    The product (as a new matrix).
     *
     * This is just straightforward matrix multiplication,
     * using the add and multiply methods of LLInt.
        * I am assuming for LLInt x, y that
     * x.add(y)      returns the sum x+y, without affecting x or y, and
     * x.multiply(y) returns the product (x)(y), without affecting x or y.
     * If that is not the case, feel free to adapt this code as required.
     */
    public Mat2x2 mul( Mat2x2 m )
    {
        return new Mat2x2(
            (a11.multiply(m.get11())).add(a12.multiply(m.get21())),
            (a11.multiply(m.get12())).add(a12.multiply(m.get22())),
            (a21.multiply(m.get11())).add(a22.multiply(m.get21())),
            (a21.multiply(m.get12())).add(a22.multiply(m.get22()))
            );
    }

    /** Compute and return the square (matrix product) of this matrix. */
    private Mat2x2 square()
    {
        return this.mul( this );
    }

    /** Compute the nth power of this matrix.
     *  This method is recursive and uses repeated squaring.
     *  @param n    The power (n>=0) to which to raise this matrix.
     *  @return    a new matrix equal to this times itself, n times.
     */
    public Mat2x2 pow( int n )
    {
        if ( n<0 )
            throw new IllegalArgumentException("neg expo");
        if ( n==0 ) {
            LLInt u0 = null, u1 = null;
            try {
                u0 = new LLInt(0);
                u1 = new LLInt(1);
            }
            catch (IllegalArgumentException e) {}
            catch (InstantiationException e) {}
            return new Mat2x2( u1, u0, u0, u1 );
        }
        if ( n==1 )            // redundant base case,
            return this;        // saves 1 recursive step
        if ( (n & 1) == 0 )
            return pow( n/2 ).square();
        else
            return mul( pow( n/2 ).square() );
    }

    public String toString()
    {
        return "[ "+a11+", "+a12+"; "+a21+", "+a22+" ]";
    }

    public Object clone()
    {
        try {
            return super.clone();
        }
        catch (CloneNotSupportedException e) {
            throw new Error("Impossible!");
        }
    }
}


/* Driver class, has a main()),
 * reads from command line and computes the answer.
 */
public class Fibonacci {

    /* Fibonacci computation */
    private static LLInt fastFibo( int n )
        throws InstantiationException
    {
        if (n < 0)
            throw new IllegalArgumentException("neg. Fibonacci");

        LLInt u0 = new LLInt(0), u1 = new LLInt(1);
        return new Mat2x2( u0,u1,u1,u1 ).pow(n).get12();
    }

    /* User interface. */
    public static void main( String[] argv )
        throws InstantiationException
    {
        int n = Integer.parseInt( argv[0] );
```

Download Print

Activity Details

You have viewed this topic

Last Visited May 18, 2016 8:38 AM