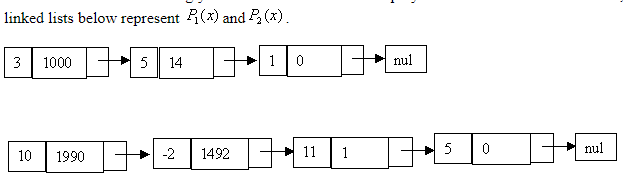
***CSC 241 Programming Assignment #5 Fall 2017***

***The Polynomial ADT***

We can define an abstract data type for single-variable polynomials (with non-negative exponents) by using a list. Let http://www.cs.rochester.edu/u/pawlicki/CSC172SP01/CSC172/Project%201_files/image002.gif. **If most of the coefficients ai are nonzero we could use a simple array to store the coefficients and write routines to perform addition, subtraction, multiplication, differentiation, and other operations on these polynomials.**

**An array implementation would be adequate for dense polynomials, where most of the terms are present,**but if http://www.cs.rochester.edu/u/pawlicki/CSC172SP01/CSC172/Project%201_files/image004.gif and http://www.cs.rochester.edu/u/pawlicki/CSC172SP01/CSC172/Project%201_files/image006.gif,  then the running time is likely to be unacceptable. One cans see that if arrays were used, then most of the time is spend multiplying zeros and stepping through what amounts to nonexistent parts of the input polynomials. This is always undesirable.

An alternative is to use a singly linked list. Each term in the polynomial is contained int one cell, and the cells are stored in decreasing order of exponents. For instance, the 

***Assignment***

Starting with the class Skeleton, below, Implement the Polynomial ADT using linked lists. You are Not to use the Java LinkedList class, you should “roll your own” list/node data types.

Your project will be graded automatically, so you must make the Polynomial ADT it’s own class. However, you must include an application that demonstrates and tests the key functionality of your implementation. Your write/documentations up should include a discussion of what you used and the documentation html files generated by **javadoc**.

public class Literal {

            // various constructors & accessors (not shown)

            double coefficient;

            int exponent;

            //if you roll your own list, then include “Literal next;”

}

public class Polynomial {

            private List terms ; // A list of literals

// if you roll your own, then “private Literal head;”

            public Polynomial() {

                        // constructor to be implemented

            }

            public void insert term(double coef, int exp){

                        // to be implemented

            }

            public Polynomial add(Polynomial rhs) {

                        // to be implemented

            }

            public Polynomial multiply(Polynomial rhs) {

                        // to be implemented

            }

            public String toString(){

                        // to be implemented use “^” to signify exponents

            }

}

***GRADING :***

Documentations 10%

Code Style 15%

Functionality 75% ( 15% for each base operation supported +.-./,\*,())

Extra Credit :do not require blanks (i.e. allow users to input "5+7\*2" as well as "5 + 7 \* 2" )  5% must be documented - it's your responsibility to point it out)

Extra Credit : support negative and positive numbers - unary +/-  (i.e.allow users to input  " -5 + -7  \*  +2"  )  5% must be documented - it's your responsibility to point it out)

Extra Credit : exponentials (use "^" operator associating right to left)  5% must be documented - it's your responsibility to point it out)

Extra Credit : modulo  (use "%" operator same association as in JAVA )  5% must be documented - it's your responsibility to point it out)