Sit-in Lab #3

CS1020 AY2010/11 Semester 2
Date: 24 March 2011, Thursday
School of Computing, National University of Singapore

Instructions

There is only **one** exercise in this lab, but with multiple parts. Time limit is **1 hour 40 minutes**.

Please do NOT reveal the question to anybody until tomorrow, 25 March 2011.

Exercise (Polynomial)

Background

This exercise is about using **SortedLinkedList** class to represent **Polynomial** class for arithmetic operations. The terms in a polynomial are ordered in decreasing order of their exponents.

Technical

A term has two parts: its coefficient, and its exponent. For example, a term of coefficient 3, exponent 13, means $3x^{13}$; while a term, for example, of coefficient 4, exponent 0, means the constant 4. For the polynomial that is zero (i.e. no term), it is represented as a SortedLinkedList whose **head** = **null**.

```
class Term implements Comparable <Term> {
    private Integer coefficient;
    private Integer exponent; // assume it is always non-negative

    public Term(int coef, int expo) {coefficient=coef; exponent=expo;}
    public Boolean isZero () { return (coefficient==0); }
    public String toString() { // code omitted };
    public int compareTo (Term x)    { // Part A: code to be filled in by you ~ 10% };
    public boolean equals (Object x) { // Part B: code to be filled in by you ~ 10%};
    public Term sum (Term B) { // Part C: code to be filled in by you ~ 10%};
    public int value (int m) { // Part D: code to be filled in by you ~ 10%};

}
```

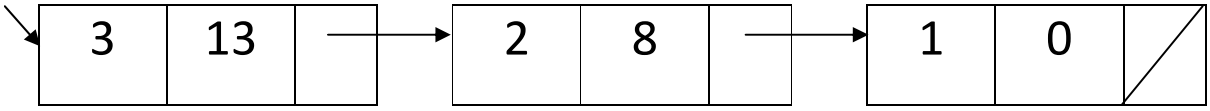
In the following, a term $3x^{13}$ is drawn in a simple manner as:

3	13

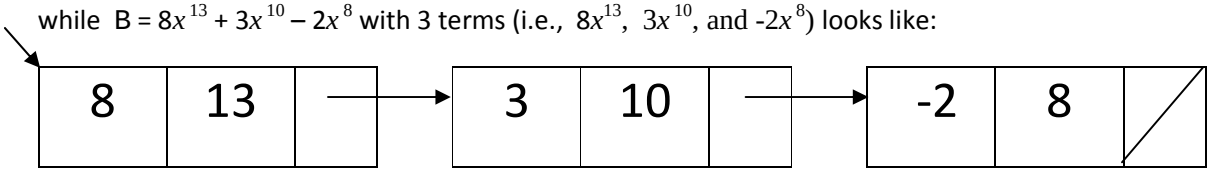
even though 3 and 13 are actually objects from the Integer class rather than primitive int.

The meaning of each method is documented in the file **Polynomial.java**.

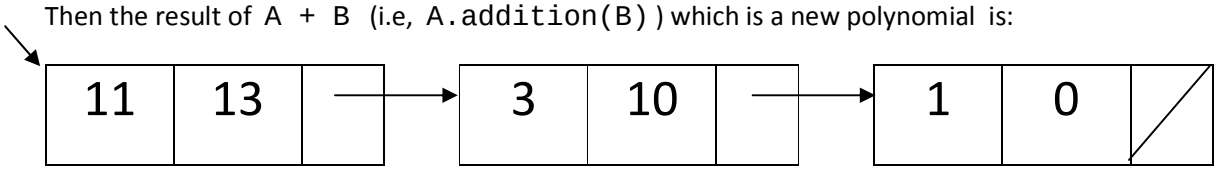
A polynomial is a sequence of terms **sorted** in decreasing order according to their exponents in a linked list (so, each node has a **term** plus a **next** pointer). For example, the polynomial A = $3x^{13} + 2x^{8} + 1$ with 3 terms (i.e., $3x^{13}$, $2x^{8}$, and constant 1) is stored as:



while B = $8x^{13} + 3x^{10} - 2x^{8}$ with 3 terms (i.e., $8x^{13}$, $3x^{10}$, and $-2x^{8}$) looks like:



Then the result of A + B (i.e, `A.addition(B)`) which is a new polynomial is:



Notice that the term with coefficient 0 for x^{8} in the result of A + B is no longer recorded for the new polynomial. The skeleton of the Polynomial class is as follows. The meaning of each method is documented in the file **Polynomial.java**.

```
class Polynomial extends SortedLinkedList <Term> {
    public String toString() { // code omitted };
    public int evaluate (int m) { // Part E: code to be filled in by you ~ 20% };
    public Polynomial addition(Polynomial polyB){//Part F: code to be filled in by you
                                                 //      ~ 40%};
}
```

Files in Directory

There are various files given in the directory "Question". You do not need to modify most of the files, except for the following:

(1) **Polynomial.java** : this is the skeleton file for you to work on. Credit will be given to only coding done to this file. Please note that you are not allowed to create new methods other than those already mentioned in the file.

(2) **TestPoly.java** : this is the driver class where you can use to test Polynomial.java. You can use the file to directly test Polynomial.java with the default 2 polynomials, or you can comment the first part and uncomment the second part to read input from keyboard for your testing purpose. At the end of the exercise, you need not worry about changes you made to the file, as this file will not be graded.

The other files in the directory are the usual linked list files that you have seen them in lecture or in tutorial or in take-home lab. In particular, **SortedLinkedList.java** is a part of the answer to our take-home lab #3.

There is another directory called "Backup" which contains all the files as you originally have, should you need them for reference still.

Restriction on Coding

You have Part A to Part D in the class Term, and Part E to Part F in the class Polynomial to complete. You should do incremental coding from a part to another part. Please observe the following highlights when you do your coding:

1. Each polynomial is created so that its terms are sorted in decreasing order of their exponents.
2. Do NOT modify any place of the given source codes that you are not asked; otherwise, penalty will be imposed on your submission. In particular, do not add (or remove) any data members or methods in **Term** class and **Polynomial** class

Sample Run

We assume no error in the input. With the default inputs in **TestPoly.java**:

```
polyA.add( new Term(2,8) );
polyA.add( new Term(3,13) );
polyA.add( new Term(1,0) );
polyB.add( new Term(-2,8) );
polyB.add( new Term(8,13) );
polyB.add( new Term(3,10) );
```

which are exactly the two polynomials drawn in previous page, the output from running: **java TestPoly** is

Polynomial A: 3x^13 + 2x^8 + 1

Polynomial B: 8x^13 + 3x^10 - 2x^8

Polynomial C: 11x^13 + 3x^10 + 1

Polynomial C(1) = 15

Marking Scheme

The allocated marks are as indicated in each part. Please note that 20% of the marks in each part is allocated to the style and design of your solution.

=== End of paper ===

Next few pages are the printout of the file Polynomial.java for ease of your reference.

```java
// CS1020 AY2010/11 Semester 2
// Sit-in Lab #3
//
// <Fill in your name and discussion group here>

import java.util.*;
import java.io.*;

////////////////////////////////////////////////////////////////////////////////
///
/// Class Term
///
////////////////////////////////////////////////////////////////////////////////

class Term implements Comparable <Term> {
    private Integer coefficient;
    private Integer exponent; // assume always non-negative

    public Term (int coef, int expo) {
        coefficient = coef;
        exponent = expo;
    }

    public boolean  isZero ( )      { return (coefficient==0); }

    public String toString () {
        if (coefficient < 0)
            if (exponent != 0) return " - " + (- coefficient) + "x^" + exponent;
            else return " - " + (- coefficient);
        else
            if (exponent != 0) return " + " + coefficient + "x^" + exponent;
            else return " + " + coefficient;
    }

    ////////////////////////////////////////////////////////////////////////////
    // PART A.  compareTo method (10%)
    //
    // The exponent of 'this' is compared with the exponent of 'x'
    // return 0  if their exponents are the same
    // return -1 if the exponent of 'this' is greater than that of 'x'
    // return 1  if the exponent of 'this' is less than that of 'x'
    //
    // IMPORTANT: If you implement this method correctly, the output from printing the
    //            polynomial should be in decreasing order of the exponent terms
    ////////////////////////////////////////////////////////////////////////////
    public int compareTo (Term x) {
        // Part A. Write your code below....

        return 0; // changes needed

        // end of Part A
    }

```

```
54      ////////////////////////////////////////////////////////////////////////////
55      // PART B.  equals method (10%)
56      //
57      // The exponent of 'this' is compared with the exponent of 'x'
58      // return true if their exponents are the same
59      // return false otherwise
60      //
61      ////////////////////////////////////////////////////////////////////////////
62      public boolean equals (Object x) {
63          // Part B. Write your code below....
64
65          return true; // changes needed
66
67          // end of Part B
68      }
69
70      ////////////////////////////////////////////////////////////////////////////
71      // PART C.  sum method (10%)
72      //
73      // The exponent of 'this' is compared with the exponent of 'x' to be sure
74      // they are the same before summing up to give the result
75      //
76      // if this.equals(B) is true, return a new term which is the sum of the input terms
77      // else return Term(0, this.exponent)
78      //
79      // E.g 'this' is 8x^4, and B is -2x^4, result is a new term 6x^4
80      // E.g 'this' is 8x^4, and B is -2x^3, result is a new term 0x^4
81      ////////////////////////////////////////////////////////////////////////////
82      public Term sum ( Term B ) {
83          // Part C. Write your code below....
84
85
86          return (new Term(0,0)); // changes needed
87
88          // end of Part C
89      }
90
91      ////////////////////////////////////////////////////////////////////////////
92      // PART D.  value method (10%)
93      //
94      // return the value of this term by substituting x by m
95      //
96      // E.g this is the term 8x^4, then this.value(2) returns 8(2^4)=128
97      ////////////////////////////////////////////////////////////////////////////
98      public int value ( int m ) {
99          // Part D. Write your code below....
100
101         return 0; // changes needed
102
103         // end of Part D
104     }
105 }
106
```

```
107 ////////////////////////////////////////////////////////////////////////////
108 /// Class Polynomial
109 ////////////////////////////////////////////////////////////////////////////
110 class Polynomial extends SortedLinkedList <Term> {
111
112     public String toString ( ) {
113         if (head == null) return "";
114         Iterator <Term> itr = iterator();
115         String str = (itr.next()).toString();
116         str = str.replaceAll("\\+","");
117         while ( itr.hasNext() ) str = str + itr.next();
118         return str;
119     }
120     ////////////////////////////////////////////////////////////////////////////
121     // PART E.  evaluate method (20%)
122     //
123     // evaluate the value of the polynomial at a given value m
124     // that is: substitute the variable 'x' with the integer value 'm'
125     // it is also the sum of value(m) of each term of the polynomial
126     ////////////////////////////////////////////////////////////////////////////
127     public int evaluate (int m) {
128
129         // Part E. Write your code below....
130
131         return 0; // changes needed
132
133         // end of Part E
134     }
135     ////////////////////////////////////////////////////////////////////////////
136     // PART F.  addition method (40%)
137     //
138     // add the 'this' polynomial with the given polyB, and
139     // return a third polynomial polyR
140     ////////////////////////////////////////////////////////////////////////////
141     public Polynomial addition (Polynomial polyB ) {
142         Polynomial polyR = new Polynomial();
143         Iterator <Term> itrThis = this.iterator();
144         Iterator <Term> itrPolyB = polyB.iterator();
145         Term A, B, R;
146
147         // Part F. Write your code below....
148         // Hint (you can ignore if not required):
149         // You need to check the exponents of the terms of the 2 polynomials
150         // Case 1: when they are the same, you just add them up....
151         // Case 2: when the term in 'this' is "larger than" that in 'polyB',
152         //           you need to insert the term from 'this' into 'polyR'
153         // Case 3: when the term in 'this' is "smaller than" that in 'polyB',
154         //           you need to insert the term from 'polyB' into 'polyR'
155         //
156         // At the end, do not forget to insert the leftover from 'this' or from 'polyB'
157
158
159         // end of Part F
160         return polyR;
161     }
162 }
"Polynomial.java" 162 lines --64%--
```