Computer Science 320SC – 2017

Assignment 4 – programming

Due: October 2 (9pm)

Requirements

This assignment lets you get familiar with simple parsing and dynamic programming algorithm design. It is worth 5% of your total course marks. You need to solve one of the following two problems.

Problem 1: Nested List Sets

We have a special type of set where the syntax is depicted by the following grammar:

```
Set          :=   "{" Elementlist "}"
Elementlist  :=   <empty> | List
List         :=   Element | Element "," List
Element      :=   Atom | Set
Atom         :=   "{" | "}" | ","
```

<empty> stands for the empty word, i.e., the list in a set can be empty.
Note the alphabet consists of the characters which are also used for the syntax of the set. At first thought one may think that it is not possible to decide efficiently if a word consisting of "{", "}" and "," is a syntactically correct representation of a set or not. However, using dynamic programming design techniques, your task is to write an efficient program that will decide this problem.

Input Specification

The first line of the input file contains a number representing the number of lines to follow.
Each line consists of a word, for which your program has to decide if it is a syntactically correct representation of a set. You may assume that each word consists of between *1* and *300* characters from the set { "{", "}", "," }.

Output Specification

Output for each test case whether the word is a set or not. Adhere to the format shown in the sample output.

Sample Input

```
4
{}
{{}}
{{}},{,}}
{,,}
```

Sample Output

```
Word #1: Set
```

```
Word #2: Set
Word #3: Set
Word #4: No Set
```

Problem 2: Maximum Evaluation (ignoring precedence)

Given an expression using only * and + binary operators and integer operands we want to "evaluate it" such that we obtain the largest possible result. For our task we do not follow the usual arithmetic precedence of doing multiplication (denoted by *) before addition (denoted by +). For example,

```
 5 + 3 * 2
```

can be evaluated as (with parentheses used to denote operator ordering):

```
(5+3)*2=16
```

or

```
5+(3*2)=11
```

Input will be one line containing a positive integer n, followed by n lines of test expressions, each with at least *1* and at most *200* operators. Each operator and operand is separated by at least one space character. For each case, output the maximum evaluation value over all possible operator orderings.

Each integer in the input expression can fit in a 32-bit word. However, note you may need to use a multi-precision data type (e.g., *BigInteger*) to express some of the answers for the harder test data.

Sample Input

```
3
5 +   3 * 2
100 * 100 + 100 * 100 * 100 + 100 * 100
-1 + -4   * -3
```

Sample Output

```
16
40000000000
15
```

What to submit

For problem 1 — for easy (E) and harder (H) input cases — the filenames must be: (setsE.ext) and (setsH.ext), where ext is your compiler extension. For problem 2 the filenames must be: (evalE.ext) and (evalH.ext).

The easy parts of the assignment are worth 2 marks and the harder parts are worth 3 marks. The maximum total marks is 5. That is, you need to do at least one of the harder parts to get full marks of 5.