Computer Programming Assignment 4

Checkpoints

1. You should do the assignment in your own. You are not allowed to share code with others and/or copy code from other resources. If you are caught, as in the syllabus, you will get a failing grade.
2. Grading will be done in the Linux environment using Java 11.
3. g++ compiler version is g++ (GCC) 8.2.1
4. Program failed to compile/run will result 0.
5. Do not loop your program to repeat unless you are told so.
6. Do not change input/output format unless you are told so.
7. Write your name and student number at top of program as a comment.
8. Do not include Korean (and any other language than English) comment. In some encoding formats, Korean comments will cause compilation errors in the Linux environment, which will result in a 0 for your grade.
9. If you have any questions, please contact cp2018ta@tcs.snu.ac.kr

Submission

1. Submit your assignment on eTL.
2. Zip your file (or tar) as '<Student ID>-assign4.zip'
    a. ex.) 2017-12345-assign4.zip
3. Due date of this assignment is Nov 30th, 2018, 23:59
4. No late submission is allowed.

Problem 1 Hanoi Tower(C++)

For the C++ implementation assignment, we will be writing a program to solve the Tower of Hanoi problem. The definition for the Tower of Hanoi can be found here: https://en.wikipedia.org/wiki/Tower_of_Hanoi

For each iteration, you must portray the position of each disk. For instance, if the number of disk is 3, it would first look like this:

```
      *          |          |
     ***         |          |
    *****        |          |
-----------------------------------
```

1st iteration:

```
      |          |          |
     ***         |          |
    *****        |          *
-----------------------------------
```

2nd iteration:

```
      |          |          |
      |          |          |
    *****       ***         *
-----------------------------------
```

And so on.

The final result would look like this:

```
      |          |          *
      |          |         ***
      |          |        *****
-----------------------------------
```

For test cases, we will not test your program for more than 8 disks. So you should adjust the space required for up to 8 disks. Make sure this part of the assignment is done in C++, not Java.

Input will be given as a standard input of a number.

Grade

```
g++ -o Hanoi Hanoi.cpp
(g++ -std=c++11 -o Hanoi Hanoi_c11.cpp)
(g++ -std=c++17 -o Hanoi Hanoi_c17.cpp)
```

Submission
Hanoi.cpp (Hanoi_c11.cpp, or Hanoi_c17.cpp)

Problem 2 Tensor(Java)

Write three java classes: **MyScalar**, **MyVector**, and **MyMatrix**. These classes must extend parent abstract class Tensor. Also, **FinalDriver** java class will be provided, but you should modify “tensor” method to read, write and call each class’ methods.

The abstract parent class **Tensor** has an abstract add, multiply, and toString method.

MyScalar, **MyVector**, and **MyMatrix** are allowed to have add with any of the Tensor’s child classes. The add method is defined as a commutative operator,

i.e., Tensor1.add(Tensor2) returns the same result as Tensor2.add(Tensor1).

vn denotes n-th dimension of vector, and Rn denotes n-th row.

- scalar1.add(scalar2) returns a new instance of **MyScalar**, where two scalar value is added.

- scalar.add(vector) or vector.add(scalar) returns a new instance of **MyVecto**r, where the scalar’s value is added to each element in the vector.
  -- s + (v1, v2, …, vn) = (s+v1, s+v2, …, s+vn)

- scalar.add(matrix) or matrix.add(scalar) returns a new instance of **MyMatrix** ( ) where scalar value is added to each element in the matrix
  -- s + (R1, R2, …, Rn) = (s+R1, s+R2, …, s+Rn)

- vector1.add(vector2) returns a new instance of **MyVector**, where vector value is added with each corresponding dimension.

- vector.add(matrix) or matrix.add(vector) returns a new instance of **MyMatrix** obtained by adding vector to each row of the matrix.
  -- V + (R1, R2, …, Rn) = (V+R1, V+R2, …, V+Rn)
  Ex)

$$\begin{matrix}1 & 2 & 3\end{matrix} + \begin{matrix}1 & 3 & 2\\ 4 & 6 & 5\end{matrix} = \begin{matrix}2 & 5 & 5\\ 5 & 8 & 8\end{matrix}$$

- matrix1.add(matrix2) returns a new instance of **MyMatrix** obtained by adding the corresponding elements in the two matrices.
  The multiply method is not a commutative operator for involving matrix.

- scalar1.multiply(scalar2) returns a new instance of **MyScalar**, where two scalar values are multiplied.

- scalar.multiply(vector) or vector.multiply(scalar) returns a new instance of **MyVector**, where the scalar’s value is multiplied with each element in the vector.
  -- s * (v1, v2, …, vn) = (s*v1, s*v2, …, s*vn)

- scalar.multiply(matrix) or matrix.multiply(scalar) returns a new instance of **MyMatrix** where scalar value is multiplied with each element in the matrix

$$\lambda\mathbf{A} = \lambda\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix} = \begin{pmatrix} \lambda A_{11} & \lambda A_{12} & \cdots & \lambda A_{1m} \\ \lambda A_{21} & \lambda A_{22} & \cdots & \lambda A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda A_{n1} & \lambda A_{n2} & \cdots & \lambda A_{nm} \end{pmatrix}$$

- v1.multiply(v2) returns a new instance of **MyScalar**, where scalar value is

$$\sum_{k=1}^{n} v1_k * v2_k$$

- vector.multiply(matrix) returns a new instance of **MyVector** obtained by

$$\sum_{k=1}^{n} v_k m_{kj}$$

- matrix.multiply(vector) returns a new instance of **MyVector** obtained by

$$\sum_{k=1}^{n} m_{ik} v_k$$

- m1.multiply(m2) returns a new instance of **MyMatrix** obtained by

$$(m1m2)_{ij} = \sum_{k=1}^{m} m1_{ik} m2_{kj}$$

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

$$\mathbf{AB} = \begin{pmatrix} (\mathbf{AB})_{11} & (\mathbf{AB})_{12} & \cdots & (\mathbf{AB})_{1p} \\ (\mathbf{AB})_{21} & (\mathbf{AB})_{22} & \cdots & (\mathbf{AB})_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ (\mathbf{AB})_{n1} & (\mathbf{AB})_{n2} & \cdots & (\mathbf{AB})_{np} \end{pmatrix}$$

Again, this operation is valid only if the number of columns in the first matrix is equal to the number of rows in the second matrix. When an operation is not valid, output should be “null”.

You may assume that dimension of vector and matrix for operation add and multiply will always be compatible so that the operation is valid.

The toString method should also be overloaded, and each child's toString() method just prints the value of itself to the file name from args[3].

For the **MyVector** class, implement a permuteCompare() method that returning a boolean value. This method allows only **MyVector** class as parameter. v1.permuteCompute(v2) returns true if the elements of v1 form a permutation of the elements of v2.

Ex) (1 2 3 4 5).permuteCompare((5 2 4 3 1)) => true
Ex) (1 2 3 4 5).permuteCompare((1 3 2 6 5)) => false
Ex) (1 1 1 2 2).permuteCompare((1 1 2 2 2)) => false
Ex) (1 1 2 2).permuteCompare((1 1 2 2 2)) => false

All the field values must be private. You might add some variables or methods if you think it is necessary.

Methods of each classes:

```
public abstract class Tensor{
     public abstract Tensor add(Tensor t);
     public abstract Tensor multiply(Tensor t);
     public abstract String toString();
}
public class MyScalar extends Tensor{
     public MyScalar(int input) {}
     public Tensor add(Tensor t) {}
     public Tensor multiply(Tensor t){}
     public String toString() {}
}
public class MyVector extends Tensor{
     public MyVector(int[] input) {}
     public boolean permuteCompare(Tensor t) {}
     public Tensor add(Tensor t) {}
     public Tensor multiply(Tensor t) {}
     public String toString() {}
}
public class MyMatrix extends Tensor{
     public MyMatrix(int[][] input) {}
     public Tensor add(Tensor t) {}
     public Tensor multiply(Tensor t) {}
     public String toString() {}
}
```

Input will be given as a **file input**.
The first line contains two numbers – we can name it as ‘N’ and ‘X’ where ‘N’ denotes the number of “abstract tensor” to be stored, and ‘X’ denotes the number of operation lines.
Following lines have a character first and size of the tensor.(0~1000)
Characters ‘S’, ‘V’, ‘M’ stand for **MyScalar**, **MyVector**, and **MyMatrix** respectively.
For ‘S’, ‘V’ and ‘M’, there will be 1 or 2 numbers indicating the size of the “abstract tensor”.(1~40000) After that, several values will follow representing each element’s value.

- MyScalar: S [value]
- MyVector: V [dimension] [val1] [val2]
- MyMatrix: M [dimension1] [dimension2] [val 1/1] [val 1/2] … [val 1/dim2] [val 2/1] … [val dim1/dim2]

After ‘N’ lines, ‘X’ calculation command lines will follow.
Calculation lines have indexes of the ‘N’ Tensors and operator signs such as ‘+’(add), ‘*’(multiply), and ‘p=’(permuteCompare).
Ex)

```
3 5
S 2             // 1
V 2 1 2         // 2
M 2 2 1 2 1 0   // 3
1 + 2           // addition of tensor 1(S 2) and tensor 2(V 2 1 2)
2 + 3
2 * 3
3 * 2
2 p= 2
```

Output: All results will be printed to a file using toString() method.
Ex2)

```
6 5
S 3
S 4
M 1 3 2 2 2
V 5 1 2 3 4 5
V 3 2 4 5
M 3 2 1 2 3 1 2 3
1 * 2
3 * 2
6 * 3
5 + 3
3 + 3
```

For **MyScalar**, you only should print a “Scalar” + value.

```
Ex)
Scalar 2
```

For **MyVector**, you should print “Vector” + each element separated by (space).

```
Ex)
Vector 1 2 3
```

For **MyMatrix**, you should print “Matrix” + each row with nextline (separated by a (space)).

```
Ex)
Matrix
0 0 1
0 1 0
1 0 0
```

In the output, input **Tensor**s are printed first before the results of operations.

```
Ex)
Scalar 2
Vector 1 2
Matrix
1 2
1 0
Vector 3 4
Matrix
2 4
2 2
Vector 3 2
Vector 5 1
True

Ex2)
Scalar 3
Scalar 4
Matrix
2 2 2
Vector 1 2 3 4 5
Vector 2 4 5
Scalar 12
Matrix
8 8 8
null
Matrix
```

4 6 7
Matrix
4 4 4

Grade

```
javac *.java
java FinalDriver [input file] [output file]
```