PLU February 2012 Programming Contest

Advanced Problems

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 10.
2. All problems have a value.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

Number	Name
Problem 1	J Box
Problem 2	Math
Problem 3	Triangles
Problem 4	Palins
Problem 5	Hook
Problem 6	Music
Problem 7	Next Permutation
Problem 8	Dedupe
Problem 9	Park
Problem 10	Snowflakes

Good luck!

1. J Box

Input File: box.dat

General Statement: Print out the j box as shown below.

Input: The first line in the data file will indicate the number of data sets to follow. Each data set will contain the size of the box to be printed. The size is an integer greater than zero.

Output: Print out the j box of the appropriate size as shown below. The boxes are separated by one blank line.

Example Input File:

```
3
3
5
4
```

Output to screen:

```
###
#J#
###

#####
#JJJ#
#JJJ#
#JJJ#
#####

####
#JJ#
#JJ#
####
```

2. Math

Input File: math.dat

General Statement : You are on planet jj and they do math a bit differently. Math is done using the following operators : @, %, and #. @ is multiply by 3. % is add 5. # is subtract 7. That is all they can do in terms of math operations. Each expression will start with a number and then a list of operators.

Input: The first line in the data file will indicate the number of data sets to follow. Each data set will contain an expression to solve.

Output: Print out the answer formatted to 2 decimal places.

Assumptions: The expression is evaluated from left to right.

Example Input File:

```
3
3 @ %
10.4 # % @
8 #
```

Output to screen:

```
14.00
25.20
1.00
```

3. Triangles

Input File: triangles.dat

General Statement: Read in a letter and a number. The number indicates how big the letter triangle should be. The number indicating the size of the triangle will have a range from 0 to 250 (i.e., num>=0 and num<=250).

Input: The first number indicates the number of data sets to follow. Each data set will contain one letter and one number. All letter input will be uppercase.

Output: Print out the letter triangles in the order given. There is one blank line between each letter triangle.

Assumptions: The letters must wrap around from Z to A. If you start with Z and have to print 5 levels, you must wrap around and start with A after the Z level is complete.

Example Input File:

```
3
5 A
3 Z
4 C
```

Output to screen:

```
A
BB
CCC
DDDD
EEEEE

Z
AA
BBB

C
DD
EEE
FFFF
```

4. Palins

Input File: palins.dat

General Statement: A palindrome is a sequence of one or more characters that reads the same from the left as from the right. Z, TOT and MADAM are palindromes, but ADAM is not. Write a program that reads a sequence of strings and for each string determines the number of UNIQUE palindromes that are substrings of the original string and outputs all of the unique palindromes.

Input: The input file consists of a number of strings (one per line). The strings being checked are one word strings that contain no spaces.

Output: Output all of the unique palindromes. Output the unique palindromes in order based on length. If two palindromes have the same length output them in order of first occurrence in the original word. For input string ADAM, the UNIQUE palindromes are A, D, M and ADA so the output would be as follows :

4 - "A" "D" "M" "ADA"
Output the unique palindromes in order of occurrence in the original word, sorted by length.

Example Input File:

```
MOM
MADAM
TOT
ADAM
```

Output to screen:

```
3 - "M" "O" "MOM"
5 - "M" "A" "D" "ADA" "MADAM"
3 - "T" "O" "TOT"
4 - "A" "D" "M" "ADA"
```

5. Hook

Input File: none

General Statement: Print out the word Hook as shown below.

Input: none

Output: Print out the word Hook as shown below.

Example Input File:
none

Output to screen:
```
#   # #### #### #   #
#### #   # #   # # #
#### #   # #   # # #
#   # #### #### #   #
```

6. Music

Input File: music.dat

General Statement: Silence is useless. We need the world to be full of music. In computer music, volume is represented as an array of integers of length n. Silence in music is defined by 2 parameters c and m. Silence occurs in any sub-array of length m where the difference between the minimum element and the maximal element is no greater than c. For example if the volume array is [1,3,2,7,5,4,6,6] and m = 3, c = 2. Then silence occurs in the subarrays of size 3 indexed at 0, 4, and 5. Output the total number of silence intervals.

Input:
The first line in the data file is an integer that represents the number of data sets to follow. Each data set is 2 lines. The first line contains n, m, c (n <= 20, m<=20, c<=1000). The second line contains n integers representing the volume array. Each integer is less than or equal to 1000.

Output:
Print the total number of silence intervals for each data set.

Example Input File:
```
3
8 3 2
1 3 2 7 5 4 6 6
4 2 1
1 2 3 4
4 5 900
1 1 1 1
```

Example Output To Screen:
```
3
3
0
```

7. Next Permutation

Input File: perm.dat

General Statement: Given an integer A, only other integers that are permutations of A are useful. All other numbers are useless. Find the next largest integer B, where the digits in B is a permutation of the digits of A. For example, suppose A=2413, then the next largest permutation is 2431. If A is already the largest permutation, output "USELESS".

Input:
The first line in the data file is an integer that represents the number of data sets to follow. Each line is a single integer A (A <= 2,000,000).

Output:
Print B, the next largest permutation. If it doesn't exist print "USELESS".

Example Input File:
```
4
237531
1234
4321
3444
```

Example Output To Screen:
```
251337
1243
USELESS
4344
```

8. Dedupe

Input File: dedupe.dat

General Statement: Redundancy in this world is pointless. Let's get rid of all redundancy. For example
AAABB
is redundant. Why not just use AB? Given a string, remove all consecutive letters that are the same.

Input:
The first line in the data file is an integer that represents the number of data sets to follow. Each data set is a single string. The length of the string is less than 100. Each string only contains uppercase alphabetical letters.

Output:
Print the deduped string.

Example Input File:
```
3
ABBBBAACC
AAAAA
ABC
```

Example Output To Screen:
```
ABAC
A
ABC
```

9. Park

Input File: park.dat

General Statement: You like looking at trees. A park has many trees. Therefore, you spend most of your waking hours at the park. You have an acute vision problem that severely limits your ability to look at these trees. The park has several paths that you can walk on. Each path is described by an infinite-length horizontal or vertical line in the 2D plane. No trees lie on a path. You want to know how many trees are visible in the park. Because of your vision problem, a tree is visible only if you can view it by standing on some path while facing in a direction perpendicular to that path. Of course all trees are equally wide, so if there is a second tree between the first tree and where you are currently standing, then you can't see the first tree. Given location of all the trees and all the paths, determine how many trees you can view in the park.

Input:
The first line in the data file is an integer that represents the number of data sets to follow. Each data set contains several lines. The first line contains two integers n and m, the number of trees and number of paths. The next n lines contains two integers x and y, which are the coordinates of the trees. The following m lines contains a line equation of the form x=a or y=a.

Output:
Print the number of trees you can view.

Example Input File:
```
2
6 3
-1 3
4 2
6 2
6 3
6 4
4 3
x=0
y=-1
y=5
1 2
2 3
x=5
y=5
```

Example Output To Screen:
```
5
1
```

10. Snowflakes

Input File: snowflakes.dat

General Statement: Snowflakes can be symmetrical both vertically and horizontally. Snowflakes that aren't symmetrical are useless. A snowflake is essentially a two dimensional array of features. A feature is a single digit between 0 and 9. Horizontal symmetry occurs if you flip the snowflake horizontally. It is still the same snowflake. Likewise, vertical symmetry is defined in the same way. A snowflake is "Beautiful" if its horizontally symmetrical, "Graceful" if its vertically symmetrical, and "Magnificent" if it is both. Otherwise it is "Useless". For each snowflake, print what kind of snowflake it is.

Input:
The first line of data contains n, the number of inputs. Each input starts with a single integer k, k <= 20, denoting the dimension of the square snowflake. Then k lines follow with each line containing a string of length k denoting the digits in the snowflake.

Output:
For each input. print the type of snowflake.

Example Input File:

```
3
3
123
321
143
2
11
11
3
121
232
343
```

Example Output To Screen:

```
Useless
Magnificent
Graceful
```