Computer Science Competition
Region 2017
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

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.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Alessandra
Problem 2	Brandon
Problem 3	Camilo
Problem 4	Denise
Problem 5	Edward
Problem 6	Fabiana
Problem 7	Honza
Problem 8	Lisa
Problem 9	Manuela
Problem 10	Rishi
Problem 11	Suhani
Problem 12	Tanvi

1. Alessandra

Program Name: Alessandra.java Test Input File: None

Alessandra just learned in algebra class about the values of E and PI, where E is the base of the natural logarithms, and PI is the ratio of the circumference of a circle to its diameter.

She also discovered that depending on how you store it in a computer program, the precision of the output is different, and decided to write a program to demonstrate that. See if you can do that with this problem.

Input: None

Output: Write a program that displays the four values you see here: two different precision values each of the math constants E and PI.

Exact Output Expected:

```
2.7182817
2.718281828459045
3.1415927
3.141592653589793
```

2. Brandon

Program Name: Brandon.java Test Input File: brandon.dat

Brandon was playing around with a program to output square values and decided to display them in either integer format or decimal format depending on the odd/even nature of the value. See if you can replicate his program.

Input: A series of integer values, all on one line, with single space separation.

Output: The square value of each number, either in integer format, or decimal format, based on the odd or even state of the number, as shown in the sample data and output below.

Sample input:
```
56 11 -34
```

Sample output:
```
3136
121.0
1156
```

3. Camilo

Program Name: Camilo.java **Test Input File: camilo.dat**

Camilo is fascinated with patterns and has come up with one that he thinks is interesting. Using a single integer value, he creates an asymmetrical arrow shape.

Study it carefully and see if you can write a program to create these shapes.

Input: Several integers N, such that 3 <= N <= 20, each on one line.

Output: For each integer N, an asymmetrical arrow shape as shown below, relative in size to the value N. A blank line will follow each arrow shape.

Sample input:
```
3
4
5
```

Sample output:
```
*
**
***
****
***
*

*
**
***
****
*****
***
*

*
**
***
****
*****
******
*****
***
*
```

4. Denise

Program Name: Denise.java **Test Input File: denise.dat**

Denise loves to go to concerts. However, every concert she goes to, she tends to drop nearly all the contents of her purse. She always realizes when it's about time to leave, but she has all her friends with her. Since it is time to go, she is in a hurry and needs your help to determine how to most quickly retrieve the contents of her purse, now scattered across the location.

Since Denise has all her friends with her, she can use as many as necessary to help her retrieve her items, but she needs to optimize how far each of her friends needs to go. Due to the crowd, oftentimes it isn't quite a straight shot between the edge of the concert grounds and her purse contents, so her friends will need to snake through the crowd. However, if on the path to retrieve another item from her purse, a friend walks past a different item, they should go ahead and pick up that item too. Therefore, you are trying to find the shortest path to each object, but, if the shortest path to an object also walks past another object, then the friend should pick up that object too.

If to grab an item two friends need to walk partially the same path and split off, that is acceptable; many of her friends can occupy the same space at the same time. The start of any of her friend's paths must be any non-obstacle space (not a #) **on the edge of the spill area**. Denise is always too traumatized and flustered when this happens, therefore never helps to pick up her own items; only her friends will pick up her spilled items.

```
####.##.##
###○....##
###...#○##
###○###○##
#o..###○##
#...###○##
#.....o..#
###.######
###..o..##
#######.##
```

```
####A##B##
###AA..B##
###A..#B##
###A###B##
#AAA###B##
#...###B##
#.....BB.#
###.######
###..CCC##
#######C##
```

The first sample data set is shown in the first box, with the solution depicted in the second box. For this situation, Denise only needs 3 friends, indicated by the letters A, B and C in this diagram, showing the path they took to retrieve the items they encountered. This is true because the shortest path to the **bold** objects provides access to all the other items.

```
##########
###....○##
###...####
###.######
#○..#....#
###.######
###.#....#
###.######
###.....##
#######.##
```

```
##########
###..BBB##
###BBB####
###B######
#AA%#....#
###%######
###%#....#
###%######
###%%%%%##
#######%##
```

In these two boxes, the situation and solution for the second data set are shown. Only two friends are needed, and as you can see, they walk the same path for a while, indicated by the % symbol starting at the bottom right of the map, finally branching off to retrieve the two items that were spilled.

Denise (cont)

Given a map of the spill area, and the location of Denise's purse contents, find the shortest path to the objects. If a shortest path to a farther item contains the most number of other items, choose that path. Determine how many such paths exist (and thus how many of Denise's friends are needed) to collect *all* her items.

Sample Input:
```
3
####.##.##
###○....##
###...#○##
###○###○##
#○..###○##
#...###○##
#.....○..#
###.######
###..○..##
#######.##
-
##########
###....○##
###...####
###.######
#○..#....#
###.######
###.#....#
###.######
###.....##
#######.##
-
##########
###.○..○##
###...####
###.######
#○..#....#
###.######
###.#....#
###.######
###....○##
#######.##
-
```
Sample Output:
```
3
2
2
```

5. Edward

Program Name: Edward.java **Test Input File: edward.dat**

Edward has found a board game similar to Go which uses Xs and Os on an 8X8 checker/chess style board. He wants to write a program to simulate the game and needs help with one part of the AI (artificial intelligence). Given a situation at some point in the game, he wants to know if certain pieces have "captured" other pieces.

For an X piece to capture an O piece, it must be adjacent to it, and have another X on the other side of the O, vertically or horizontally, but not diagonally, also adjacent to it.

For example, in the board shown below, the O at position (6,3) is captured by the Xs at positions (5,3) and (7,3). The same is true for the O at position (6,6), captured by the Xs at positions (6,5) and (6,7). However, the two Os in the bottom right corner at positions (1,8) and (2,8) are not captured horizontally since they are on the edge, even though they are surrounded by X pieces. However, a piece that is on the edge can be captured if there are X pieces on either side alongside of it. For example, if position (1,8) was an X piece, the O piece in position (2,8) would be captured.

```
--------
--X-----
--O-XOX-
--X-----
--------
-------X
------XO
------XO
```

What Edward needs to know is when the position of an X position is indicated, what O piece, if any, is it helping to capture? For example, if the X piece at position (7,3) is indicated, the answer would be the position of the O piece it is helping to capture, which would be (6,3). If the X piece at position (1,7) is mentioned, there is no O piece it is helping to capture, so the answer would be NONE. Since this step is just the starting point in the overall AI of the game, Edward will only use data where at most one O is captured by any X piece. Once he perfects that, he'll move on to multiple possibilities, but not now.

Input: An initial value N, indicating N data sets to follow. Each data set consists of two rows, one for the O pieces, and the next for the X pieces. Each row starts with a value Q, followed by Q ordered coordinate pairs (x followed by y) indicating the positions of the pieces. After the two rows of pieces, an integer P indicates P ordered pairs, each on one line, representing the X pieces to query.

Output: The coordinates of the given X piece and of the O piece it is helping to capture, or the phrase "NONE" if there is no capture, exactly as shown and formatted in the examples below. It is guaranteed that each X piece will help capture at most one O piece. There is exactly one space on either side of the arrow for each output. A blank line will follow each complete output.

Sample Input:

```
2
4 1 8 2 8 6 3 6 6
7 1 7 2 7 3 8 5 3 7 3 6 5 6 7
5
7 3
6 5
5 3
3 8
1 7
4 1 1 2 2 2 6 1 7
9 2 1 1 2 3 2 1 5 2 5 2 3 1 6 2 7 1 8
6
3 2
2 1
1 2
1 5
1 8
2 7
```

Sample Output:

```
(7,3) ==> (6,3)
(6,5) ==> (6,6)
(5,3) ==> (6,3)
(3,8) ==> NONE
(1,7) ==> NONE

(3,2) ==> (2,2)
(2,1) ==> (2,2)
(1,2) ==> (2,2)
(1,5) ==> NONE
(1,8) ==> (1,7)
(2,7) ==> (2,6)
```

6. Fabiana

Program Name: Fabiana.java **Test Input File: fabiana.dat**

Fabiana has just learned about the depth concept as it relates to binary search trees and decides to do some research. She takes a single word, picks a letter from somewhere in the word, and finds the depth of where that letter settles into a binary search tree. Doing it by hand is not difficult, but is time consuming, and she needs your help in writing a program to do it for her to save her some time doing the research.

Given a single word with no spaces, followed by an integer P after a blank space, output the depth at which the letter in position P is inserted into a binary search tree.

Below is an example of a binary search tree formed by using the letters in the word REGIONUIL. The first letter is always the root of the tree, and then each subsequent letter "finds" its place relative to the root, and any other nodes in the tree, going to the left if its value is less than or equal to the current node, and going right if it is greater in value.

Depth levels
```
0    R
    / \
1  E   U
    \
2    G
      \
3      I
      / \
4    I   O
        /
5      N
      /
6    L
```

The root node that contains the letter R has a depth of zero, with the letters E and U at level 1, G at level 2, and so on.

The data set consisting of the word REGIONUIL and the value 0 means to find where the letter at position 0, the R, settles in the tree, which would be the root, or level 0.

The data set REGIONUIL 7 indicates the second occurrence of the letter I, which is inserted into the tree at level 4 as shown.

Input: A series of data sets, each on a separate line, consisting of a word (all uppercase) and an integer.
Note: The judges test data input will have different words than the one given here.

Output: The depth (or level) at which the indicated letter in the word settles in the binary search tree.

Sample Input:
```
REGIONUIL 0
REGIONUIL 1
REGIONUIL 3
REGIONUIL 5
REGIONUIL 7
```

Sample Output:
```
0
1
3
5
4
```

7. Honza

Program Name: Honza.java **Test Input File: honza.dat**

Honza is a time traveler. He has been visiting different psychics all throughout time and space, to answer one single question, when will he meet his soulmate! All the psychics give their answers in the form of minutes from the current date and time.

Honza's time machine uses the standard numeric settings for the current date and time in the form month/day/year, followed by time of day in military format, such as 4/7/2017 17:00, indicating 5 pm on April 7, 2017. It also can calculate the day of the week for the given date and reports it in abbreviated form, such as Mon for Monday, Tue for Tuesday, and so on with Wed, Thu, Fri, Sat, and Sun.

Given the time and date, and the number of minutes reported by the psychic, determine the day and time that Honza will need to set his time machine to be on time to meet his soul mate. Since he is a stickler for being on time, and firmly believes that "on time is late", he arrives five minutes early, just to be sure.

Honza's psychics will only report times in the future. However, if necessary, Honza can travel back in time in order to be "on time" to meet his soul mate.

For example, using the first data set shown below, Honza is told by the psychic at one minute before the stroke of midnight on December 31, 1999 that he will meet his soulmate in 7 minutes, therefore he decides to be five minutes early and arrives 2 minutes later Saturday, January 1, 2000, at 1 minute past midnight.

Input: The first line, N, will be the number of data sets to follow. Each data set will have three items: the date, the current time, and the number of minutes to wait, formatted as shown below. *Note: Honza may have to wait a very, very long time.*

Output: Display the 3 letter abbreviation of the day of the week, followed by the date in month/day/year format followed by the time in hour:minute format.

Sample Input:
```
3
12/31/1999 23:59 7
12/31/1993 00:32 9193343326845736209
2/1/523 02:59 1000000000
```

Sample Output:
```
Sat 1/1/2000 00:01
Sun 6/6/2951 11:46
Fri 5/31/2424 14:34
```

8. Lisa

Program Name: Lisa.java Test Input File: lisa.dat

Lisa has decided to play around with her friends, well, with their names, anyway. She has just learned about how different letters of the alphabet have integer values that represent them. For example, the uppercase letter 'A' has a value of 65, and little 'a' has a value of 97. The values for 'B and 'b' are 66 and 98, and the rest of the alphabet follows in sequence.

She decides to calculate a "name weight" for each person, which consists of a complex formula. The formula takes three sums, averages those three sums, and then divides the average by the number of letters in each person's name. The first sum is found by adding up the ASCII values for each person's name, just as it is, with the first letter in uppercase, and the rest in lower case. The second sum adds up all the ASCII values of the uppercased version of the name, and the third sum adds up the values for the lowercased version.

Using her own name as an example, the values for characters in the name **"Lisa"** are 76, 105, 115 and 97, which add up to 393. For the uppercased name, **"LISA"**, the values are 76, 73, 83 and 65, for a total of 297. The lowercased name, **"lisa"**, has ASCII values of 108, 105, 115 and 97, totaling 425. These three sums are added together... 393+297+425 ... to equal 1115, which is then divided by 3 to equal 371.66667, which is then divided by the length of her name, 4, to make 92.916667.

The output for each name is to include the original form of the name, the three sums in order – original, uppercased, lowercased – and the final "name weight" average, formatted to two decimal places.

Finally, all the names need to be output in descending order by the final "name weight" average, as shown below.

Input: Several names of friends, each on one line.

Output: All the names in descending order by "name weight" average, as described above and shown in the example below. The alignment must match EXACTLY as shown. It is guaranteed that the judge's test data will have no names longer than those contained in this sample data. The name "Alessandra" is the longest name in the data set, and occupies ten spaces, with is one space between her name and the following number. The rest of the data is guaranteed to fit accordingly.

Sample Input:
```
Lisa
Alessandra
Brandon
Camilo
Denise
```

Sample Output:
```
Alessandra 1022  734 1054 93.67
Brandon     708  516  740 93.52
Lisa        393  297  425 92.92
Denise      600  440  632 92.89
Camilo      597  437  629 92.39
```

9. Manuela

Program Name: Manuela.java **Test Input File: manuela.dat**

Manuela is a space traveler. She regularly flies around the ‘verse in her ship, a Jupiter-wing class 45X-328A. However, in all her travels, she is getting increasingly frustrated by all the various currencies she keeps having to carry around. As such, she would like your help to determine what is the best way to make change in all the different places she goes.

Given the names and the values of the currency of the place where she is arriving, and the amount she needs to make change for, tell her how much of each coin she should need to use the *least* number of coins.

Input: The first line, N, will be the number of data sets to follow. Each data set will begin with a number, i, representing the number of unique types of coins. The next i lines will be the name of each coin, followed by its value. After the ith line, there will be a final value representing the amount of change that needs to be made.

Output: Display the coins used, in alphabetical order, followed by a colon, and the number of each coin needed to use the minimum number of coins.

Sample Input:
```
3
6
two-dollar 200
dollar 100
quarter 25
dime 10
nickel 5
penny 1
337
7
aijika 13
uilika 11
quilika 7
yuilkia 5
dilkia 3
gilikia 2
tikki 1
14159
6
euro 100
pentuplepenny 5
quadruplepenny 4
triplepenny 3
doublepenny 2
penny 1
1027
```

Sample Output:
```
dime: 1 dollar: 1 penny: 2 quarter: 1 two-dollar: 1
aijika: 1089 gilikia: 1
doublepenny: 1 euro: 10 pentuplepenny: 5
```

10. Rishi

Program Name: Rishi.java **Test Input File: rishi.dat**

Data compression is new to Rishi in his computer study, and he is fascinated with it. He has been experimenting with different methods and needs your help with a process called run-length encoding, which simply counts sequences of like data and creates a shorter string in a coded fashion.

For example, the string in the first sample data item below has several instances of repeated letters, which can be encoded into a shorter string by counting the number of instances of the letter and listing that letter once, preceded by the number of occurrences of that letter.

As you can see, the letter 'A' occurs 4 times at the start, which results in the coded string "4A". That is followed by 4 Bs, 2 As, 6 Cs, and 7 Ds, resulting in the encoded string shown.

Decompressing such a string is a simple task as well. You just work through the encoded string and expand it, as shown in the second example. The first number/letter pair is 12X, which means "XXXXXXXXXXXX", followed by "YYYY" and "ZZZZZZZZZZZZ".

Input: A string to be compressed, or a compressed string to be decompressed. The string to be compressed will contain only uppercase letters, and the encoded string will contain a series of number/letter pairs.

Output: The resulting decompressed, or compressed, string from the given input.

Sample Input:
```
AAAABBBBAACCCCCCDDDDDDD
12X4Y12Z
RSRSRSRRRRRRRRSSSSSSSSSSSSSST
6W3B9W4B3W
```

Sample Output:
```
4A4B2A6C7D
XXXXXXXXXXXXYYYYZZZZZZZZZZZZ
1R1S1R1S1R1S8R14S1T
WWWWWWBBBWWWWWWWWWBBBBWWW
```

11. Suhani

Program Name: Suhani.java **Test Input File: suhani.dat**

In geometry class, Suhani is studying about congruent triangles and has learned that triangles are congruent when all three side lengths of one triangle match in some way the three sides of the other triangle. For example, a triangle ABC with side lengths 3, 4, and 5 is congruent to another triangle DEF with side lengths of 3, 4 and 5. Side A matches with side D, B with E, and C with F.

Even if the lengths are listed in a different order, like 3, 4 and 5 for the triangle ABC and then 4, 5 and 3 for DEF, the two triangles are still considered congruent. Side A matches F, B matches D, and C matches E.

If not all three sides match, it is still possible for some of the sides to match, or none at all.

Suhani wants to be able to take the side lengths of any two triangles and determine how many side lengths match, which could be none, 1, 2, or all 3.

Input: Several sets of data, each set on one line, consisting of six non-negative values, with single space separation.

Output: The number of sides that match between the two triangles.

Sample Input:
```
3 4 5 3 4 5
2 3 4 4 3 2
3 4 5 5 2 3
2 3 4 2 3 4.5
2 2 3 3.5 2 2
```

Sample Output:
```
3
3
2
2
2
```

12. Tanvi

Program Name: Tanvi.java **Test Input File: tanvi.dat**

Tanvi is excited about her schooling. Unfortunately, her excitement for learning has caused her to sign up for too many courses this year. Her course load has gotten so heavy that she can't even attend all her classes in a day because many of them occur at the same time or are overlapping in some way. Thus, she needs your help to determine the maximum number of classes she can attend in a day.

Please help make Tanvi's agenda for the day of the classes she can attend, starting the beginning of each day, zero hour (00:00). None of her classes will ever cross over into the next day. Each of her classes must begin strictly after the previous one has ended.

Input: The first line, N, will be the number of data sets to follow. Each data set will begin with a number, i, representing the number of classes in this day. The following line will be a list of i times, in the format [HH:MM, HH:MM] representing when the class begins and ends.

Output: In the format shown in the sample output below, print the classes that Tanvi can make so that she attends the most number of classes during the day. If she can make more than one class in a certain time slot, prefer the one that ends earlier (not necessarily the shortest).

Sample Input:
```
3
5
[00:00, 13:15] [10:45, 11:00] [10:00, 10:31] [11:01, 13:50] [23:00, 23:10]
6
[17:00, 22:30] [01:45, 03:33] [01:00, 02:50] [01:45, 03:00] [02:45, 03:01] [11:05, 11:09]
5
[22:30, 23:30] [01:00, 02:50] [02:45, 03:00] [17:00, 22:29] [02:51, 04:42]
```

Sample Output:
```
[10:00, 10:31] [10:45, 11:00] [11:01, 13:50] [23:00, 23:10]
[01:00, 02:50] [11:05, 11:09] [17:00, 22:30]
[01:00, 02:50] [02:51, 04:42] [17:00, 22:29] [22:30, 23:30]
```