Computer Science Competition
2017 District
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	Alexey
Problem 2	Beverly
Problem 3	Dong-Sun
Problem 4	Gan
Problem 5	Hyo
Problem 6	Judy
Problem 7	Kate
Problem 8	Michael
Problem 9	Neelam
Problem 10	Pratik
Problem 11	Ralph
Problem 12	Suresh

1. Alexey

Program Name: Alexey.java Input File: alexey.dat

Alexey's cousin Isidora recently shared with her a new concept she had learned in CS class called bit string flicking, which included operations like **left shift, right shift, left circle,** and **right circle** operations. Alexey was fascinated with this new idea and decided to do some further research on it, and discovered other operations that were used with bit string manipulation, like **NOT**, **AND**, **OR**, and **XOR**, which she knew worked with logic operations, but now she discovered could also work with binary strings.

She continued using Isidora's notation of using **RS** for Right Shift, **LS** for Left Shift, **RC** for Right Circle, and **LC** for Left Circle as the first part of the command structure, with a slight adaption of using a single space instead a dash, a value, another single space, and then the base ten integer value to be shifted or circulated. She added **N** for Not, **A** for AND, **O** for OR, and **X** for XOR, each preceding the operands, with single space separation.

As she had learned from Isidora, the shift command **RS 4 45** will take the value 45, convert it to the binary string **101101**, and then perform a **Right Shift 4**, which means the rightmost 4 bits are eliminated, leaving 10 in binary as the result, which is equivalent to the base ten value **2**. The command **LS 2 13** converts the value 13 to its binary equivalent of **1101**, and performs a **Left Shift 2** operation, which essentially appends two zeroes to the back of the binary string, resulting in **110100**, or **52** base ten.

The circle commands result in a same size binary string as the original, with a Right Circle taking the rightmost bits and circling them to the other side, and likewise for the Left Circle, taking the leftmost digits and circling them to the other side. For example, the command **RC 3 81** will take the binary value for 81, or **1010001** in binary, take the three right most digits, 001, and circle them to the front of the string resulting in **0011010,** which is **26** in decimal.

The four new commands would work like this.

- **N 23** would take the binary equivalent for 23, which is 10111, and "flip" all of the bits, resulting in 01000, or just 1000, which is equivalent to the decimal integer value **8**.
- The prefix style expression **A 23 14** would perform the bitwise AND operation on the two values 23 and 14, which in binary would be 10111 AND 1110, resulting in 00110, or **6**.
- The expression **O 23 14** is equivalent to 10111 OR 1110, which results in 11111, or **31**.
- The expression **X 23 14** is equivalent to 10111 XOR 1110, which has a result of 11001, or **25** in base ten.

Input: Several commands as described above, each on one line. It is guaranteed that the command will work within the length of the string, with all results greater than or equal to zero.

Output: The resulting base ten value for the command.

Sample input:
```
RS 4 45
LS 2 13
RC 3 81
N 23
A 23 14
O 23 14
X 23 14
```
Sample output:
```
2
52
26
8
6
31
25
```

2. Beverly

Program Name: Beverly.java Input File: none

Beverly has just started learning how to output in Java and just wants to write a program to say hello. To show here how it's done, write a simple program to say hello to the judges, with a creative, fun, and respectful salutation, to get them on your good side, something like, **`"Top of the mornin' to you, kind judges!"`** or **`"We appreciate you judging us today, and know you will do a great job!"`**

Input: None

Output: A single sentence, all on one line, containing an **original**, creative, fun, and respectful greeting to the judges.

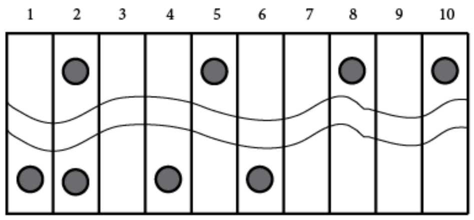
Sample input:
None

Sample output:
```
Top of the mornin' to you, kind judges!
```

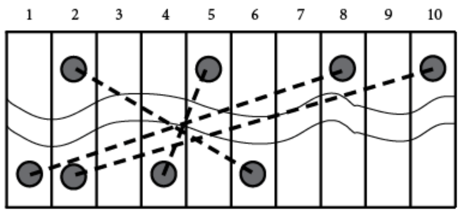
3. Dong-Sun

Program Name: DongSun.java   Input File: dongsun.dat

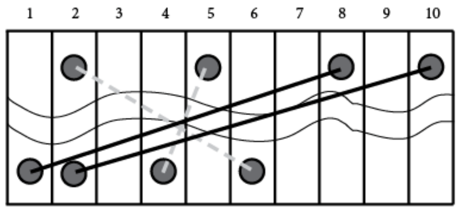
Dong-Sun is currently employed as a civil engineer building bridges in an area with an east-west running river and many cities on either side. These cities all have a lot of traffic between them, and many have filed bridge applications with his office. However, when looking through the applications, he's noticed that some have been filed that would intersect, which cannot be done, for obvious reasons.
The cities in Dong-Sun's area are labelled by their position along the north or south bank of the east-west running river. For example, here is cities 2, 5, 8, 10 on the north bank; and 1, 2, 4, 6 on the south bank:



The bridge applications that Dong-Sun has are bridges from city 10 to 2, 5 to 4, 2 to 6 and 8 to 1, These proposed bridges are indicated by the following notation, **`10 -> 2, 5 -> 4, 2 -> 6, 8 -> 1`**, with the first city of each notation pair always the one on the north side of the river, and the second one always on the south side. It is guaranteed that any proposed bridge will cross the river. The two north/south cities at position two tried to submit a `2 -> 2` bridge proposal, but missed the deadline.



Dong-Sun cannot build all these bridges because he can't approve bridges that intersect, but he does want to approve as many as he can. In this scenario, the most number of bridge applications he can approve is two, the **`10 -> 2`** bridge and the **`8 -> 1`** bridge, since these are the only ones that can be built without any bridges intersecting.



Given the bridge applications that Dong-Sun receives, in the notation from ***`north side city -> south side city`***, determine which bridges he should approve in order to provide the maximum number of bridges built for use by the cities.

Input: The first line, N, will be the number of data sets to follow. The first line of each data set will be the number of applications, I, which Dong-Sun has received. The next I lines will be the bridge applications in the form ***`north side city -> south side city`***, where ***`north side city`*** and ***`south side city`*** will be integers representing their location along the river bank.

Output: For each data set, output the applications that Dong-Sun can approve, in the order they were initially given. The output should be prefaced by "Approved Applications: " followed by a comma separated list of the approved bridge applications.

Dong-Sun - Sample input:

```
3
4
10 -> 2
5 -> 4
2 -> 6
8 -> 1
10
3 -> 4
1 -> 2
2 -> 3
6 -> 7
10 -> 1
8 -> 9
4 -> 5
9 -> 10
5 -> 6
7 -> 8
6
1 -> 15
2 -> 8
8 -> 3
9 -> 2
10 -> 2
11 -> 15
```

Sample output:

```
Approved Applications: 10 -> 2, 8 -> 1
Approved Applications: 3 -> 4, 1 -> 2, 2 -> 3, 6 -> 7, 8 -> 9, 4
-> 5, 9 -> 10, 5 -> 6, 7 -> 8
Approved Applications: 9 -> 2, 10 -> 2, 11 -> 15
```

4. Gan

Program Name: Gan.java Input File: gan.dat

Gan is doing some work with circles in his math class, specifically calculating circumference and area, along with surface area and volume of spheres. Using the standard formulas of $2\pi r$, πr^2, $4\pi r^2$, $4/3\pi r^3$, and using the same radius value for all four expressions, his homework is to calculate and express to a precision of one decimal point, the outcomes for all of these calculations, and needs your help to write a program to check his work, showing, in a nice clear, well-aligned format, the results of the calculations. He decides to use the value of PI, precise to 15 places, which is 3.141592653589793.

Input: Several integers, all on one line, with single space separation.

Output: For each integer, output the original value as the radius of a circle, followed by the circumference, area, surface area, and sphere volume using the given radius, formatted exactly as shown below, with dash lines above and below the output, arranged, formatted and aligned exactly as shown below.

Sample input:
```
2 5 3 12
```

Sample output:
```
------------------------------------------
 2     12.6     12.6     50.3     33.5
 5     31.4     78.5    314.2    523.6
 3     18.8     28.3    113.1    113.1
12     75.4    452.4   1809.6   7238.2
------------------------------------------
```

5. Hyo

Program Name: Hyo.java Input File: hyo.dat

Hyo is researching Facebook friend recommendations. He wants to see if there is a pattern in the degree of separation between two people and the Facebook friend recommendations. The degree of separation between two people is defined by the number of friendship steps between them. Two people who are friends directly with each other have a separation degree of 1.

For example:
Carla has 3 friends, Jeff, Martin, and Adrian.
Jeff has 1 friend, Carla.
Martin has 2 friends, Carla, and Sam.
Adrian has 1 friend, Carla.
Sam has 1 friend, Martin.

The degree of separation between Carla and Sam is 2, because Carla is a friend of Martin who is a friend of Sam. In this same scenario of friendships, the degree of separation between Jeff and Carla is 1. One degree of separation can be expressed using the phrase, **`"is a friend of"`**, so it can be said that **`"Jeff is a friend of Carla"`**.

Given a scenario of Facebook friendships, and two people within that scenario, determine the degree of separation of the two people, and the friendship steps connecting them.

Input: The first line, N, will be the number of data sets to follow. Each data set will begin with an integer K, denoting the number of people in the friendship scenario. Each scenario will begin with a single name followed by a colon. All names after the colon represent direct friends of that person, separated by single spaces. The final line will have two names that indicate the two people for which their degree of separation and their friendship steps are to be determined.

Output: For each data set, there will be two lines of output. The first line will be the sentence: **`"The degree of separation of`** ***`<person1>`*** **`and`** ***`<person2>`*** **`is`** ***`<degree of separation>`*** **`. "`** where <person1> and <person2> are the two names listed in the final line of the data set, and <degree of separation> is the number of steps between them in the friendship scenario, as defined above. The second line contains the actual friendship steps, starting with <person1> and ending with <person2>, with the string **`"is a friend of"`** separating each step. Each sentence is ended with a period.

Sample input:
```
3
3
Annie: Bahi
Bahi: Carthik Annie
Carthik: Bahi
Annie Carthik
6
Salem: Charlie Maona Blaze Cram
Charlie: Maona Joana Salem
Maona: Charlie Salem
Blaze: Salem Cram
Cram: Salem Blaze
Joana: Charlie
Salem Maona
11
Arya: Cher Ying
Erik: Qa
Cher: Arya Bo
Qi: Shome Thor
Ying: Arya Bo
Qa: Thor Erik
Bo: Ying Cher Yoshie
Yoshie: Bo Pradip
Pradip: Yoshie Shome
Shome: Pradip Qi
Thor: Qi Qa
Erik Arya
```

Hyo - Sample output:

```
The degree of separation between Annie and Carthik is 2.
Annie is a friend of Bahi is a friend of Carthik.
The degree of separation between Salem and Maona is 1.
Salem is a friend of Maona.
The degree of separation between Erik and Arya is 9.
Erik is a friend of Qa is a friend of Thor is a friend of Qi is a friend of
Shome is a friend of Pradip is a friend of Yoshie is a friend of Bo is a
friend of Ying is a friend of Arya.
```

6. Judy

Program Name: judy.java Input File: judy.dat

Judy has just learned that an array can be used to store a binary tree data structure, where the root node is stored in position 0, and the left and right children are stored in positions 1 and 2. The left and right children of the root's left child are in positions 3 and 4, with positions 5 and 6 containing the two children of the root's right child. This goes on and on, with the left child always in a node based on its parent's position * 2 + 1, and the right child in parent's position * 2 + 2.

Consider the tree below and the array diagram and study how this works. The root C is in position zero, and its left child J is in position (0 * 2 + 1), or 1, and the right child D is in position (0 * 2 + 2), or 2. The children of M, which is in position 6, are in positions 13 (6 * 2 + 1) and 14 (6 * 2 + 2).

```
    C
   / \
  J   D
 / \   \
K   L   M
       / \
      N   A
```

The resulting array representation for this is shown here:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
C J D K L – M – – –  –  –  –  N  A
```

To challenge herself, Judy decided to list on a line of data each parent/left child/right child triple as a cluster of three characters, with the parent of the triple listed first, followed by the left child, and then the right child. The root triple in the diagram would be **`CJD`**. The rest of the triples would include **`JKL, D-M,`** and **`MNA.`** The dash means there is no child in that position.

The challenge would be to list the triples in random order, put them all on one line in a data file, with a single space between each triple, and then write a program to sort it all out, putting the triples into their proper positions in the array representation of the resulting binary tree.

To test this challenge, she always included the node **`A`** once somewhere in the tree, and then tried to figure in what position it ended up in the array. For example, if the line of triples was in this order **`- MNA CJD JKL D-M`** – she would first need to find the root triple, but she discovered that was easy since the parent letter of the root triple doesn't appear in the 2nd or 3rd position of any other triple. The C of the triple CJD does not appear in any of the other triples, and so it is clearly the root node. The rest of the tree is fairly easy to figure out, working down from the root.

Input: Several data sets, each on one line consisting of a series of binary tree triples, in no certain order.

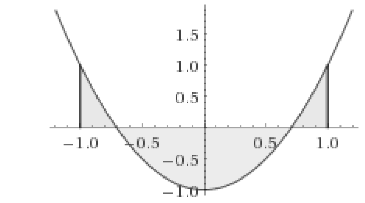
Output: The final position of the node **`A`** in the array representation of the resulting binary tree.

Sample input:
```
MNA CJD JKL D-M
SAL ATR
HB- BA- ADE
```
Sample output:
```
14
1
3
```

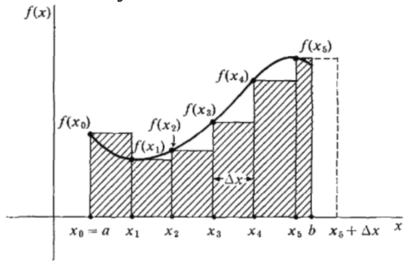
7. Kate

Program Name: Kate.java Input File: kate.dat

Kate is taking her first calculus class and is very excited to learn about integration. She understands this process takes a function and finds the area that the graph of this function makes with the x-axis. For example, the diagram below shows the integral of the function y = $2x^2$-1, with a y-axis range from –1 to 1, and a final integral value of -0.67. She asks why the answer is negative, thinking that area cannot be negative, and is answered by her teacher Ms. Sno that a negative result means that in this case, most of the area of the curve bounded by the x axis is below that axis, which made sense to her.



Ms. Sno went on to explain that integrals, when done by a computer, have to be done in a particular way; in other words, the space has to be "discretized". This means separating the x-axis into small pieces. In this process, the function is evaluated at every point, and a rectangle of the height of the function and the width of the discretion size is created. These rectangles are summed to represent the approximate value of the integral. This technique is often called a Riemann sum, named after the 19th century German mathematician Bernhard Riemann.



Kate needs your help in writing a program that simulates an integral calculator using this method. Given a function, a lower bound, an upper bound, and a discretation size, find the approximate integral value. To find the integral, start at the lower bound, and create rectangles of width equal to the discretation size and height equal to the function value evaluated at that point.

Input: The first line, N, will be the number of data sets to follow. Each data set has two lines. The first line is the equation. The equation is always a polynomial. The second line contains 3 doubles. The first double is the lower bound of the integral evaluation. The second is the upper bound of the integral evaluation. The last is the discretation size for dividing the x-axis. *Note: the given discretation size is guaranteed to always divide evenly along the distance between the upper bound and lower bound.*

Output: For each data set, output the integral value, formatted to 2 decimal places.

Sample input:
```
3
1.0x^5 + 2.2x^3 + 3.23x^4 + 4 - 3.2x
-1.4 1.6 0.1
9.323x^3 - 9x + 3
1 2 0.5
2x^2 - 1
-1 1 0.001
```

Sample output:
```
22.91
12.14
-0.67
```

8. Michael

Program Name: michael.java **Input File: michael.dat**

Michael has decided to invent a new game for his chess board called Quintos. In a way it is similar to Tetris, but with three major differences. Each ***"quinto"*** is made up of five squares, not four, and in the game, overlapping is allowed. Also, they don't rotate, but retain the same orientation. His idea is for the object of the game to be the person to cover the last square on the board, but he's still working out the finer points of the rules. In the meantime, Michael needs help with a program that shows how many squares have been covered after a certain number of pieces have been played.

He has decided on three basic shapes, A, B and C, and has selected one of the five squares in each shape to be the reference cell, for the purpose of data representation, with the other four cells in relative position to that reference cell.

The A style quinto is L-shaped, with the reference cell R being the top of the L, and rest of the squares in this shape: **R** **A** **AAA**	The B style quinto has a reference cell R also in the top left position of the overall shape, and looks like this: **RB** **BB** **B**	The C style quinto looks like an L lying in its side, like this: `   C` **CCCR**

To test his program so far, he has decided to use data in a certain format, with an initial single digit integer value N to represent how many pieces have been played, and then N sets of data, with each set consisting of the letter A, B or C, indicating the shape of the letter, and then two integers X and Y indicating the position of the reference cell of that piece. He wants to know how many of the 64 squares on the board have been covered so far by the pieces.

The reference cell for each piece is located at the (X,Y) position of the chess board, just as you would find in the first quadrant of a coordinate plane. The test data will indicate how many pieces have been played, followed by the shape and coordinates of each piece.

For example, the input line: **`4 A 2 5 C 5 7 B 7 6 C 8 1`**
produces a grid like this, with a total of 20 squares covered by quintos:

```
....C...
.CCCR...
......RB
.R....BB
.A....B.
.AAA....
.......C
....CCCR
```

If one more piece is played, say A 4 3, the board now looks like this, with some overlapping caused by the new piece, and only two additional squares actually covered by the new piece, for a total of 22 squares covered.

```
....C...
.CCCR...
......RB
.R....BB
.A....B.
.AAR....
...A...C
...AAACR
```

Michael - Input: Several data sets, each on one line consisting of single digit integers and the letters A, B and C. The first integer indicates how many "quintos" are represented on each line. Each quinto is represented by the letter A, B, or C, followed by 2 integers indicating the position of the reference cell of the quinto. It is guaranteed that all of the quintos represented by this data are fully on the board, with no part of any piece out of bounds.

Output: An integer value representing how many cells in the 8X8 grid are covered by the quintos for that data set.

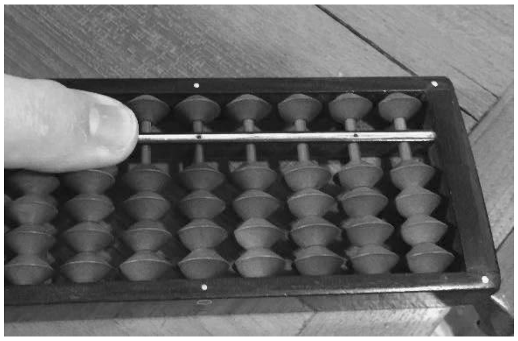
Sample input:
```
5 A 2 5 C 5 7 B 7 6 C 8 1 A 4 3
6 A 3 7 B 3 7 C 4 7 B 5 5 C 6 5 B 5 6
6 A 4 4 C 6 2 B 4 6 B 7 3 C 8 4 A 6 4
3 A 5 5 B 5 5 C 5 5
```
Sample output:
```
22
16
21
11
```

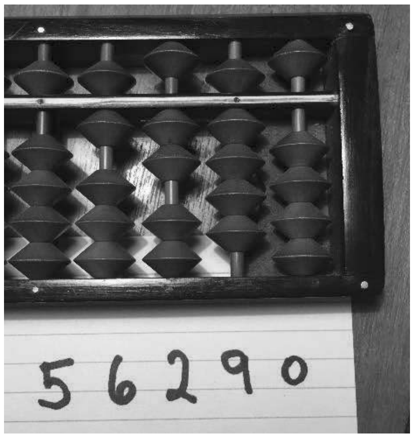
9. Neelam

Program Name: Neelam.java **Input File: neelam.dat**

Neelam has been learning about the soroban, a Japanese version of the Chinese abacus, and wants to write a program to visually express different values as they would appear on the soroban. The ancient calculating instrument consists of several rows of beads that slide on rods in two main sections. The top section only has one bead per rod, and represents the value 5. The bottom section has four beads per rod that can slide up or down as needed. There is a middle bar that divides the two sections, and the user of the soroban slides one finger along the bar from one end to the other to "clear" the calculator, forcing all beads to the outside, which represents the value zero.

On the right is a picture of what this would look like.





To express a value, in each row you move as many beads as you need to create the value, as you can see in the picture on the left.
For the value 5, the top bead is moved against the middle bar. For 6, one bottom row bead joins the top row bead against the middle bar, and adds up to 6. Study carefully how 2, 9 and 0 are represented in the same manner.

Neelam needs help visualizing a value graphically, and would like your help to do this by writing a program to show the soroban display using keyboard characters. Using the lower-case letter 'o' for a bead, the equals sign for the middle bar, and the pipe '|' symbol for the gaps, write a program to input a value and then display it in soroban format, like this for the value 56290:

```
56290
||o|o
oo|o|
=====
|ooo|
o|ooo
oo|oo
ooooo
ooo|o
```

Input: Several integers values N, 0<N<100000, each on one line.

Output: For each integer value N, display original input integer and the soroban version of that value, as demonstrated above, showing only the significant columns for the value.

Sample input:
```
56290
4371
802
```

Neelam - Sample output:

```
56290
||o|o
oo|o|
=====
|ooo|
o|ooo
oo|oo
ooooo
ooo|o

4371
oo|o
||o|
====
oooo
ooo|
oo|o
o|oo
|ooo

802
|oo
o||
===
o|o
ooo
oo|
|oo
ooo
```

10. Pratik

Program Name: Pratik.java   Input File: pratik.dat

Pratik lives in Texas, but has family members spread out all across the world. His Aunt Tapasi lives in Kolkata, India, his grandpa Aadi lives in Nepal, and his second-cousin Bongi lives in Cape Town, South Africa. Cousin Gulasyl lives in Aktobe, Kazahkstan, and Uncle Ardem lives in Ukraine!

Right now, the only member of his family that lives near him is his mother. Because of his international family, Pratik is finding it more and more difficult to keep up with everyone's whereabouts, and because of the different time zones, determining a suitable time of day to call and visit. Pratik wants to keep in touch with his family, and is asking for your help.

Given Pratik's current time of day, who resides in the US Central time zone, equivalent to GMT-6, and which family member he is trying to call, render for him a digital clock, in 24-hour time, representing the time it is for that family member.

Here are the lists of time zones for his family members:

- Cape Town, South Africa: GMT+2
- Kolkata, India: GMT+5:30
- Nepal: GMT+5:45
- Ukraine: GMT+2
- Aktobe, Kazahkstan: GMT+5

The only members of his family that he will ever try to call are: Aunt Tapasi, Grandpa Aadi, Second Cousin Bongi, Cousin Gulasyl, Uncle Ardem, and Mom.

Input: The first line, N, will be the number of data sets to follow. Each data set has two lines. The first line is the time for Pratik, given in 12 hour time, followed by an 'am' or a 'pm'. The second line is the family member he is intending to call.

Output: For each data set, output the time for the given family member, as a digital clock, in 24-hour time. Each digit of the 24-hour clock is 5x5 characters and made up of '#', except the colon separating the number, which is made of periods. There should be a new line between each data set for separation.

Sample input:
```
3
4:53 pm
Aunt Tapasi
8:59 am
Second Cousin Bongi
6:07 pm
Mom
```

Sample output:
```
##### #   #   ##### #####
#   # #   # .     #     #
#   # #####   ##### #####
#   #     # . #         #
#####     #   ##### #####

    # #####   ##### #####
    # #   # . #   #     #
    # #####   #   #     #
    # #   # . #   #     #
    # #####   #####     #

    # #####   ##### #####
    # #     . #     #   #
    # #####   ##### #####
    # #   # .     #     #
    # #####   ##### #####
```

11. Ralph

Program Name: Ralph.java **Input File: ralph.dat**

Data storage limits have always been fascinating to Ralph and he wanted to know if there was a way to test whether or not a number could be "morphed" into the largest possible value to fit into a particular data type. He knew that the maximum value a 16-bit signed value was 2^15-1, or 32767, and wanted to test his theory. He needs help to write a program to do this, given any five-digit value greater than 9999 and less than 100,000.

For example, is there a way to maximize the value 12345 by rearranging the digits to find the largest possible value that would fit in a 16-bit signed integer memory without exceeding the maximum value? He worked out possible combinations and decided that 32541 was the best combination that represented the largest possible value without exceeding the limit.

Input: Several five-digit integers N, each within the range 9999<N<100,000.

Output: The largest possible value made up of the digits in the given integer, or the message "`EXCEEDS MAX VALUE`" (significant digits only in final output).

Sample input:
```
12345
32677
33445
60405
```
Sample output:
```
32541
32767
EXCEEDS MAX VALUE
6540
```

12. Suresh

Program Name: Suresh.java **Input File: suresh.dat**

Suresh loves reading and every once in a while will do some analysis of what he is reading, just for fun.
He'll take a paragraph, count the number of sentences, the number of words in the sentence, and find the longest words in the sentence. To streamline this process, he wants a program that will do all of this work for him, and needs your help to write it.

He has decided the easiest thing to do is to type in the entire paragraph on one line in a data file, and then put another paragraph on the next line, and so on.

The output will report three different things per paragraph:

- The number of sentences in the paragraph
- The number of words in each sentence of the paragraph
- The longest word in each sentence. If there are two or more "longest words", he wants the one that occurs last in the sentence.

Input: Several paragraphs, each all on one line, containing at least one sentence, each sentence ended with a period or question mark. Each sentence contains at least two words, each word ended by a space or punctuation mark, including commas, colons, or semi-colons.

Output: For each paragraph, output the number of sentences in the paragraph, and for each sentence, the number of words, followed by the longest word for each sentence, as described above, with single space separation between each part of the output.

Sample input: (each paragraph will be all on one line in the data file)

To be or not to be, that's the question.
A wise man is not one who knows all the answers. But one who knows how to find the answers. A foolish man is one who knows neither.
Did he ever return? No, he never returned and his fate is still unknown. He may ride forever neath the streets of Boston, he's the man who never returned.
Oh, somewhere in this favored land the sun is shining bright, the band is playing somewhere, and somewhere hearts are light. And somewhere men are laughing, and little children shout; but there is no joy in Mudville. Mighty Casey has struck out.

Sample output:

```
1 9 question
3 11 answers 9 answers 8 neither
```

(Note: In the last sentence of this paragraph, the word “foolish” is the same length, but “neither” is later in the sentence.)

```
3 4 return 10 returned 15 returned
3 21 somewhere 16 somewhere 5 struck
```