Thirteenth Annual
University of Central Florida

ACM · UPE

High School Programming Tournament

Problems

Problem Name	**Filename**
These Aren't The Droids We're Looking For!	DROIDS
Ants	ANTS
Hatching Lizards	EGGS
Eugene And The Supremes	RATS
Wheebop!	ROBOTS
The Y-to-K Problem	YTOK
Dinner And A Movie	MOVIE
Inspector Doohickey	MAZE
Too Many Keys	KEYS
Round And Round We Go	FORCE

Call your program file: *Filename*.PAS, *Filename*.C or *Filename*.CPP
Call your input file: *Filename*.IN

For example, if you are solving Too Many Keys:
Call your program file: KEYS.PAS, KEYS.C or KEYS.CPP
Call your input file: KEYS.IN

These Aren’t The Droids We’re Looking For

Filename: `DROIDS`

Luke and his uncle Owen are looking for a few droids to help out on the farm. In particular, they need droids that can communicate properly with their moisture vaporators in the fields. Unfortunately, the vaporators are finicky, and will only interface with certain droids. They will only communicate with droids whose names follow certain criteria.

The Problem:

The rules are as follows:

1. The second letter of the droids name must be a number or a hyphen “-“
2. The last letter of the droids name must be a number or a vowel (“A”, “E”, “I”, “O”, “U”, or “Y”)
3. There can be no group of letters larger than two. Letter groups are separated by numbers or hyphens
4. No characters other than letters, numbers, and hyphens are permitted

Fortunately, there is a group of Jawas (little people with shiny eyes and brown robes) that scavenge and sell droids in the area where Luke and Owen live. Unfortunately, the Jawas carry droids of all shapes and sizes, and no one can tell if they’ll be able to interface with the vaporators just by looking at them. Since Luke is a much better pilot than he is a programmer, he’s asked you to write him a program that will help him and his uncle find the droids they need.

The Input:

Input will begin with a single positive integer, *N*, on the first line, indicating how many droids the Jawas have today. On the next *N* lines will be the names of the Jawas’ droids. Droid names will have at least 1 character and no more than 70. They will also contain no white space or non-printable characters.

The Output:

Output will be Owen’s response to each droid, either “`We’ll take this one.`” or “`Hey! What are you trying to push on us?`” List one droid per line, and leave one blank line after the output for each droid. Print the name of the droid, then a colon, then two spaces, then Owen’s response.

Sample Input:

```
6
R2-D2
DE-34A
J2REG3
C-3PO
A3-B&C7
G4-AB3e
```

Sample Output:

```
R2-D2:  We’ll take this one.

DE-34A:  Hey! What are you trying to push on us?

J2REG3:  Hey! What are you trying to push on us?

C-3PO:  We’ll take this one.

A3-B&C7:  Hey! What are you trying to push on us?

G4-AB3e:  We’ll take this one.
```

Ants

Filename: `ANTS`

Artificial life researcher Cantrell Montauvon is designing artificial ants. She has a demonstration of her ants scheduled with an important (and very secretive) government agency. She wants her ants to perform well, but, to be honest, they aren't nearly ready yet. However, Dr. Montauvon knows that this secretive government agency will suggest that her ants walk some triangles (secretive government agencies like triangles—no one knows why). She also knows that they are going to try to trip up her ants by giving them patterns that are not valid triangles.

So, she wants you to write a program that will analyze the input that the secretive government agency men feed to her ants, and tell the ants if the pattern could be a triangle, or if the agents are trying to trick them. Also, since the ants are especially efficient at making right-angle turns, it would be helpful to know if the pattern they are given represents a right triangle.

The Problem:

As input, the ants take three numbers representing lengths. If these lengths could possibly form a triangle, the ants can traverse them on their own. All you have to do is determine whether those three lengths could be the sides of a triangle or right triangle.

The Input:

The first line of the input file contains a single positive integer, *N*, denoting the number of data sets. Following that are *N* lines, each containing three integers separated by white space; each of these lines contains one set of lengths to be tested. All integers are between 1 and 32000 inclusive.

The Output:

Each input set must produce exactly one line of output, chosen from:

```
A right triangle can be constructed.
A triangle can be constructed.
No triangle can be constructed.
```

Sample Input:

```
3
5   6   7
2   3   10
120 50  130
```

Sample Output:

```
A triangle can be constructed.
No triangle can be constructed.
A right triangle can be constructed.
```

Hatching Lizards

Filename: `EGGS`

The hatching rate for a clutch of lizard eggs depends directly on how many eggs are in the clutch. If there are up to 50 eggs, the hatching rate is 55%. That is, only 55% of the eggs will hatch. If there are between 51 and 100 eggs, 70% of them will hatch. If there are more than 100 eggs in the clutch, 20% of them will hatch.

Once the eggs have hatched, however, some baby lizards may still die. The survival rate of hatchling lizards is related to the season in which they hatched. Lizards that successfully hatch in the spring have a 75% survival rate. That is, 75% of the lizards that hatch will survive. Lizards that hatch in summer have a 50% survival rate; lizards that hatch in fall have a 25% survival rate; and lizards that hatch in winter have only a 10% survival rate.

Now, these rates apply only to lizard eggs in a natural state. In captivity, the owner of the eggs may choose to use an incubator. An incubator does not effect the hatching rate of the eggs, but it does affect the survival rate once the eggs have hatched. The use of an incubator causes 90% of the newly hatched lizards to survive, regardless of the season in which they hatched.

The Problem:

Your task is to figure out how many baby lizards will survive in a clutch based on the size of the clutch and the season in which the clutch hatches.

The Input:

The input file starts with a positive integer, *N,* indicating the number of clutches to be evaluated. This integer is on a line by itself.

The next *N* lines contains information about each clutch to be evaluated, each on a line by itself. Each line contains a positive integer indicating the size of the clutch (number of eggs) and a string indicating the season. The size of the clutch and the season are separated by exactly one space. The season string is exactly one of these four strings (without the quotes): “spring”, “summer”, “fall”, “winter”. The season strings are guaranteed to be all lowercase characters. An asterisk immediately after the season string (not separated from the string by any white space) indicates the use of an incubator.

The Output:

For each clutch, output the clutch number (starting at 1) and the number of surviving lizards. Since a part of an egg makes no sense, round down if the number of hatching eggs is not a whole number. Likewise, since part of a lizard does not survive very well, always round this figure down as well if necessary. Follow the format of the sample output.

Sample Input:

```
5
50 spring
127 summer
19 winter
100 fall
250 winter*
```

Sample Output:

```
Clutch #1: 20 surviving lizards
Clutch #2: 12 surviving lizards
Clutch #3: 1 surviving lizards
Clutch #4: 17 surviving lizards
Clutch #5: 45 surviving lizards
```

Eugene And The Supremes

Filename: `RATS`

Eugene is a parking garage attendant in downtown Orlando. The particular garage he works at (I won't say which one) has a rat infestation. As Eugene spends most of his time working there, and it's not a very popular garage, he's had the opportunity to observe their society, and he's written down the following rules:

1. Every rat has a reputation. This reputation can be represented by an integer, and it goes up and down based on the rat's deeds and the actions of the other rats.
2. The rat whose reputation has the highest absolute value is referred to as the "Supreme Rat", and holds a privileged position in rat society.
3. When rat A slanders rat B, rat A's reputation is increased by one, and B's reputation is decreased by three *unless rat B is the Supreme Rat*. If rat B is the Supreme Rat, the rat A's reputation decreases by two, and rat B's reputation remains the same.
4. When rat A lauds rat B, rat A's reputation is decreased by one, and B's reputation is increased by three *unless rat B is the Supreme Rat*. If rat B is the Supreme Rat, the rat A's reputation increases by two, and rat B's reputation remains the same.

The Problem:

Eugene wants you to write a program to keep track of the ins and outs of rat society.

The Input:

There will be multiple data sets. The first line of the input file will contain a single positive integer, N, representing the number of data sets in the file. Each data set will begin with a single integer, M ($0 < M \leq 30$), where M represents the number of rats in Eugene's garage. The next M lines will contain the rat's names, one per line. The names will be no longer than fifteen characters and will contain only letters. The letters will all be uppercase. The rest of the data set will contain the actions of the rats, and commands to your program. Each rat's reputation starts at zero. Each line will be one of the following:

1. *Rat1* SLANDER *Rat2*
   Follow the rules above.
2. *Rat1* LAUD *Rat2*
   Follow the rules above.
3. SUPREME
   Print the name of the Supreme Rat followed by " is the Supreme Rat". In case of a tie, print "There is no Supreme Rat".
4. PRINT
   Print the names of all the rats and their reputations, in order from worst to best. If two rats have the same reputation, print them in alphabetical order.
5. END
   The end of the input set.

In each case, *Rat1* and *Rat2* are rat's names. No rat will be named SLANDER, LAUD, SUPREME, PRINT, or END. When processing a SLANDER or LAUD command, you can assume that there will be exactly one space on either side of the word SLANDER or LAUD.

The Output:

The output will be the output from the SUPREME and PRINT statements in the input. Follow the format of the Sample Output.

Sample Input:

```
1
6
ALFRED
BERTHA
CHARLIE
DENISE
EDWARD
FRANCIS
ALFRED SLANDER DENISE
BERTHA LAUD DENISE
SUPREME
FRANCIS LAUD CHARLIE
SUPREME
PRINT
EDWARD LAUD ALFRED
SUPREME
PRINT
END
```

Sample Output:

```
DENISE is the Supreme Rat
There is no Supreme Rat
DENISE: -3
FRANCIS: -1
EDWARD: 0
ALFRED: 1
BERTHA: 2
CHARLIE: 3
ALFRED is the Supreme Rat
DENISE: -3
EDWARD: -1
FRANCIS: -1
BERTHA: 2
CHARLIE: 3
ALFRED: 4
```

Wheebop!

Filename: `ROBOTS`

Due to a terrible accident involving a bottle of shampoo and a vial of acid, your friend's skull has melted, revealing his brain. It turns out, however, that this didn't hurt him at all, as he is a robot. His brain is a Pentium 166 running Linux. As he explains the intricacies of being a robot, you wonder how many other people you know are actually robots. Listening carefully, you hear the key that you need to discern who is a robot and who isn't: robots have glitches in their speech systems.

- ➔ If there are eight consecutive words in a sentence that each contain an “a” or an “e” (or an “A” or an “E”), then the eighth word is reversed. The capitalization is retained, however, so any capitalized letters in the word remain capitalized when the word is reversed.
- ➔ If the sentence has exactly one word, the robots always say "Wheebop!" instead of the sentence.
- ➔ If the sentence has more than 10 words, the 11th word is replaced with the original first word of the sentence, the 12th word is replaced with the original second word, the 13th word is replaced with the original third word, the 21st word is replaced by the original 11th word, etc.

The Problem:

Your program is to read in a file of normal text, and print it out as a robot would say it.

For the purposes of this problem, a "word" is defined as a sequence of characters separated by a space, an end-of-line, or a sentence-ender (see below).

A sentence is defined as zero or more words ended by a sentence-ender. The sentence-enders are either a period (.), exclamation (!), or question mark (?). These punctuation marks always indicate the end of a sentence, except for a few special cases of the period: "Mr.", "Mrs.", "Ms.", "Dr.", "Sr.", and "Jr." are considered words (if they are followed by a space, an end-of-line, or sentence-ender), and the period that ends the abbreviation is not considered an end of the sentence.

The Input:

The input will be a file of text. There will be no more than 70 characters per line of input.
No sentence will contain more than 100 words. There will be no tab characters in the input file.

The Output:

The output should be the input, with the above rules applied. Do not remove excess white space or carriage returns; the text should be exactly the same as the input except for the application of the above rules.

Sample Input:

```
These are the times that trie mens' feet, but not very much.
"Mom," asked Bobby Joe, "Why do you hate me so?"
"Sleep, you
foolish little boy, you are a monster and you’re Dr. Talon!" said
Mom.
```

Sample Output:

```
These are the times that trie mens' ,teef but not These are.
"Mom," asked Bobby Joe, "Why do you hate me so?"
"Sleep, you
foolish little boy, you are a monster " "Sleep, you foolish!"
said Mom.
```

The Y-to-K Problem

Filename: `YTOK`

One day at work, you receive an urgent memo from your pointy-haired boss. After a paragraph or two of mindless babbling, you realize that he's concerned about the Y2K problem. "That's nonsense," you think to yourself, "I certified all of the computers on this floor three months ago." Nevertheless, being the loyal employee that you are, you seek out your boss and question him.

"I'm worried about our computers," he says. "They're all going to crash on January first."

"No they won't," you patiently explain. "I've tested all of them, we don't have a single problem."

"Yes we do. There are still Y's everywhere. They need to be K's, right?"

The Problem:

After many long and tiring arguments, you know the futility of trying to convince your boss when he has ideas like this. So you're just going to "fix" his computer, and he'll never notice that the rest of the computers actually run. Since it's almost quitting time, you're just going to write a program that takes all of the documents on his machine and changes all the Y's to K's.

The Input:

The input will be the document that needs to be converted. You may assume that no line of text will be longer than eighty characters. Input will be terminated by end-of-file.

The Output:

The output should be the same as the input file, with all of the Y’s converted to K’s. A capital “Y” should be changed to a capital “K”, and a lower-case “y” should be changed to a lower-case “k”. No other changes should be made to the file.

Sample Input:

```
Boss's Diary

12-18-99 Things are getting very hectic around the office with
the new year coming on us.  I've got a programmer working on the
Y2K problem, so at least I have that out of my pointy hair.

12-22-99 I just finished firing a few employees before the
holidays.  Time for another wonderful christmas with the in-laws.
I saw people buying guns in preparation for Y2K on the news this
morning.  I realize that a computer crash can be annoying, but a
couple of Y's is no excuse to turn to violence.

01-01-00 Hooray, a new year for me to fire incompetent employees.
I don't think Y2K was such a big deal.  I came back to the office
this morning, and everything was in pristine order.
```

Sample Output:

```
Boss's Diark

12-18-99 Things are getting verk hectic around the office with
the new kear coming on us.  I've got a programmer working on the
K2K problem, so at least I have that out of mk pointk hair.

12-22-99 I just finished firing a few emplokees before the
holidaks.  Time for another wonderful christmas with the in-laws.
I saw people buking guns in preparation for K2K on the news this
morning.  I realize that a computer crash can be annoking, but a
couple of K's is no excuse to turn to violence.

01-01-00 Hoorak, a new kear for me to fire incompetent emplokees.
I don't think K2K was such a big deal.  I came back to the office
this morning, and everkthing was in pristine order.
```

Dinner And A Movie

Filename: `MOVIE`

Bill and Ted love to watch movies. They go to the theater every Friday night to see that week's blockbuster. They also have a few restaurants where they like to eat dinner on movie night. Unfortunately neither of them ever have the foresight to check the movie times or plan for the waiting time at the restaurant, so they never know how much time they have to get to the movie after eating dinner.

The Problem:

Since Bill and Ted are tired of getting to the theater twenty minutes after the movie has just started, they've asked you to write them a program to help them out. The program will gather movie show times and the waiting time at that week's restaurant of choice, and let them know how much time they have to get to the earliest showing possible after dinner. It always takes Bill and Ted 57 minutes to eat. Disregard travel times between work, the restaurant, and the theater.

The Input:

There will be multiple data sets. The input will start with a positive integer on the first line representing the number of data sets. Each data set will consist of three lines. The first line of each data set will be the time Bill and Ted leave work in 24-hour format. The second line will be a single positive integer representing the waiting time at the restaurant, in minutes. The third line will contain the show times of the movie in 24-hour format, each separated by a single space. Movie times will always be confined to the same 24-hour interval. The first movie of the day will not start before 11:00 am (11:00) or after 11:00 pm (23:00). Movie times will be in chronological order. There will always be at least one movie that Bill and Ted can make. No input line will be longer than 70 characters.

The Output:

For each data set, print a single line telling Bill and Ted the time of the earliest showing they can make followed by a comma, a space, and how much time they have to get there. Additionally, if they have less than twenty minutes to make it, print another comma, space, and the message "`hurry up!`" at the end of the line. Leave one blank line after each output line. See the Sample Output for the correct format.

Sample Input:

```
4
17:00
35
13:00 15:30 18:00 20:30 23:00 1:30
20:30
60
12:30 15:00 16:30 17:40 19:10 20:20 21:50 22:40 0:10 1:40
18:02
31
13:00 15:30 18:00 20:30 23:00 1:30
17:07
85
12:30 15:30 18:30 21:30 0:30
```

Sample Output:

```
20:30, you have 1 hour and 58 minutes

22:40, you have 13 minutes, hurry up!

19:30, you have 1 hour

21:30, you have 2 hours and 1 minute
```

Inspector Doohickey

Filename: `MAZE`

Dr. Talon has captured Inspector Doohickey in a maze deep beneath the earth. As the inspector's niece, Nickel, you must use your computer book to guide your cybernetically-enhanced dog, Liver, to your uncle. Liver is guided by a sequence of cardinal directions (north, south, east, and west). However, it gets a bit more complicated. The inspector believes that Liver is an evil spy, so after Liver reaches the inspector, he must lead the inspector out of the maze by allowing himself to be chased.

The Problem:

The maze is a 10x10 grid. Each spot in the grid will either be an empty space, a wall, the Inspector, Liver, or an exit. You must give a list of directions that will guide Liver from his starting point to the spot the Inspector is in and then out one of the exits.

The Input:

The input contains multiple mazes. The first line of input will contain a single integer, *N*, denoting the number of mazes to be processed. Each maze is represented by a sequence of 10 lines with exactly 10 characters on each line. The first character on the first line is the northwest corner of the maze. The last character in the 10th line is the southeast corner. The characters in each line will be one of the following:

`.`	Empty space
`W`	Wall
`L`	Liver's starting space
`I`	Inspector Doohickey
`X`	Exit spot

There will be exactly one `L` and one `I` in each maze. There will be at least one `X`.

The Output:

For each maze, print "`Maze #`*X*", where *X* is the number of the maze (starting at 1 for the first maze). On the next line, print the directions needed to get Liver to the Inspector. On the following line, print the directions to get Liver to one of the exit spots. If Liver must walk north one step, print "`N`". If Liver must walk south, print "`S`". If Liver must walk east, print "`E`". If Liver must walk west, print "`W`". Liver cannot walk through walls or out of the 10x10 area defined by the maze. Liver can, however, walk through exit spots on his way to the inspector. Do not put any spaces or other characters on the output line in addition to the directions. Print a blank line between the output for each maze. Your solution to each maze must be no longer than 200 steps where each step is a movement of exactly one square in the direction specified.

Sample Input:

```
2
WW....WWWW
WWIWW.WWWW
WW.WW.WWWW
WW.W.....W
WWWW.WWW..
X....WWWW.
WWWWWWWWW.
..L...W...
.WWWWWW.W.
........W.
WWWWWWWWWL
.....WWW.X
.WWW.WWW.W
.WWW......
.WWW.WWWW.
.X......W.
W.WWWWW.W.
W....IWW..
WWWWW..W.W
WWWWWW...W
```

Sample Output:

```
Maze #1
WWSSEEEEEEEENNEENNNWNWWWNNNWWWS
NEEESSSWSSWWWW

Maze #2
SWSSESSSSWSSWWNWN
WWWWNN
```

Too Many Keys

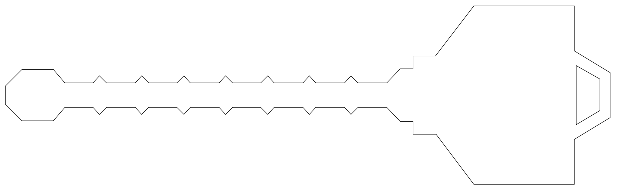
Filename: KEYS

The Upscale Condominium of Foozle is switching to a new system of making keys for its condos. This new system allows for a "Master" key while still being able to prevent tenants from opening doors other than their own. However, due to an error when the keys were made, some tenants CAN open the doors of other tenants. Management wants to find out which keys open more than one door.

The Problem:

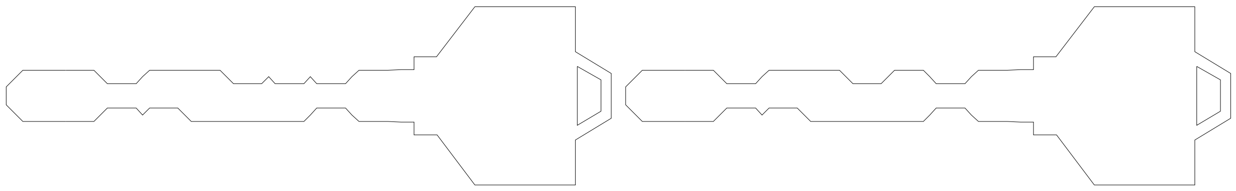
The keys have eight notch sites, each of which can be uncut, cut on the bottom, cut on the top, or cut on both the bottom and the top. (The keys are not reversible.) The cuts, if present, are always the same depth. Each lock is designed to be opened by a particular pattern of cuts. The “correct” key is the key with exactly that pattern of cuts. However, a key with too many cuts can open the locks, so long as the required cuts are present.

On the Master Key, all eight notch sites are cut on both the bottom and the top, for a key that looks like this:



A key will open a door if the required cuts are present, and since the Master Key has ALL cuts, it opens all doors.

Now consider the two keys below. The second key cannot open the door designed for the first key. The second key lacks a cut on top of the sixth site from the left. However, the first key COULD open the door designed for the second key. The first key has all the cuts the second key does, plus one more, and so it can open the “wrong” door.



The Input:

The input will consist of multiple sets. Each set begins with a non-negative integer, *N*, which indicates the number of tenants in the set ($0 \le N \le 1000$). A set with $N = 0$ indicates the end of input, and should not be processed.

Input for each tenant is on four lines. The first line will contain the name of the tenant. This name will not be longer than 15 characters and will not contain spaces. The next three lines will contain a picture of the tenant's key. No line will be longer than 20 characters. The second line of the picture will start with a `=#` and end with `######O`. There will be exactly eight characters in between which represent the notch sites. These characters will be one of the following:
"`-`" represents a notch site cut on both the top and the bottom
"`.`" represents a notch only cut on the top
"`"`" represents a notch only cut on the bottom
"#" represents a notch site that hasn't been cut at all.

The Output:

For each input set, output a set header as shown in the sample output. For each tenant in the set, output their name (starting in the third column), a colon, and then the number of other tenants' doors that the tenant can open with their key (starting in column 20). Output the tenants in the order they appear in the input. Assume that the tenant's door has been designed to be opened by the pattern of cuts on the tenant's key. Follow the output for each set with a blank line.

Sample Input:

```
3
Master
           /###\
=#--------######O
           \###/
Jones
           /###\
=##-"#..-#######O
           \###/
Smith
           /###\
=##-"#.#-#######O
           \###/
0
```

Sample Output:

```
Set #1:
  Master:          2
  Jones:           1
  Smith:           0
```

Round And Round We Go

Filename: `FORCE`

Ali was just selected as a new astronaut at NASA. As part of his training, he will be tested in the famous NASA Centrifuge. He is concerned about what kind of force he will feel in the machine so he wants you to help him figure it out. Before each test, Ali will time how long it takes the centrifuge to circle once and then you must figure out the force he will feel. The equation for the force used in this case is:

$$Force = \frac{mass \times speed^2}{radius}$$

The mass is in kilograms (kg), speed is in meters per second (m/s), and the radius is in meters (m). Force then ends up being in kilogram-meters per second squared (kg•m/s^2). We know that Ali's mass is 70 kilograms and the radius of the centrifuge is 10 meters. In order to compute the speed of the centrifuge, simply take the circumference of the centrifuge and divide it by Ali's timing.

$$Speed = \frac{2 \times \pi \times radius}{time}$$

The radius is in meters (m) and time is in seconds (s). Use a value of 3.141592654 for π.

The Problem:

Given the timing of the centrifuge, calculate the force that Ali will feel.

The Input:

The first line of the input will contain a single positive integer, *N*. On each of the next *N* lines, there will be a single data set. Each data set will consist of a single real number representing the number of seconds that it took for the centrifuge to go around once.

The Output:

For each data set, output the force that Ali will feel (rounded to three decimal places). Leave a blank line after the output for each set.

Sample Input:

```
2
1.15
3.3
```

Sample Output:

```
20895.949

2537.639
```

Fruit Loops

Filename: `LOOPS`

Johnny sat down one summer morning to a bowl of alphabet soup (you thought I was going to say cereal, right?). He began playing with the letters, spelling the names of his favorite fruits like ORANGE and APPLE and BANANA. Then he noticed that some of the letters in the name of each type of fruit had loops in them—a kind of "fruit loops."

The letter B has two loops, the letters A, D, O, P, Q, and R have one loop each, and the letters C, E, F, G, H, I, J, K, L, M, N, S, T, U, V, W, X, Y, Z have no loops.

The Problem:

Given some letters that supposedly represent the name of some kind of fruit, count the number of loops in that name.

The Input:

The first line of the input file will contain a single positive integer, *N*, which represents the number of test cases in that input file. *N* will be less than 200.

Each of the next *N* lines of the input file will contain a test case. A test case will consist only of a single "fruit" name--a sequence of one to twelve uppercase letters.

The Output:

For each test case in the input, output a single line that follows the following format:

fruit-name `has` *number* `loops.`

where *fruit-name* is the input sequence of letters and *number* is the total number of loops in those same letters.

Sample Input:

```
4
ORANGE
BANANA
LETTUCE
PLUM
```

Sample Output:

```
ORANGE has 3 loops.
BANANA has 5 loops.
LETTUCE has 0 loops.
PLUM has 1 loops.
```

Limited Productivity

Filename: `LIMITS`

Suppose you were going to write a program that takes two numbers, *A* and *B*, and multiplies them together to get the number *C*. Simple enough, right? But we're not done yet. Let's suppose that *A* and *B* both have upper and lower limits on what they can be.

As a result of the limits of *A* and *B*, their product *C* will also be limited.

The Problem:

Given the upper and lower limits for the two integer variables *A* and *B*, determine the highest and lowest possible values for what their product *C* can be.

The Input:

The first line of the input file will contain a single positive integer, *N*, which represents the number of test cases in that input file. *N* will be less than 200.

Each of the next *N* lines of the input file will contain a test case: four positive integers, each separated from the next by a single space. The numbers will be--in order--the lower limit for *A*, the upper limit for *A*, the lower limit for *B*, and the upper limit for *B*. The upper limits of *A* and *B* will be greater than their respective lower limits. All integers will be less than 100.

The Output:

For each test case in the input, output a single line that follows the following format:

The product of [A_{lower}..A_{upper}] and [B_{lower}..B_{upper}] is limited to the range [$C_{minimum}$..$C_{maximum}$].

Sample Input:

```
2
3 18 1 70
32 36 5 7
```

Sample Output:

```
The product of [3..18] and [1..70] is limited to the range [3..1260].
The product of [32..36] and [5..7] is limited to the range [160..252].
```