University Interscholastic League

Computer Science Competition

2015 Invitational A Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

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. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

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.

II. Table of Contents

1. Alien

Program Name: Alien.java Input File: none

It's your best friend's birthday, and you are throwing him a surprise party! Your friend is a big fan of old, text-based video games from the 80s (the 1980s), and you have decided to theme the party accordingly. You've never thrown a themed party before, so you and some other creative compatriots get together and put together some ideas. For the front of the invitations, you settle on the design shown below:

Input
There is no input for this problem.

Output
You are to output the ASCII-art alien shown below. Each character is of equal size (including spaces). Your output may look wrong and still be correct (it requires a fixed-width font to look the same as below), so make sure you are careful to follow the design exactly. You may assume that there are no trailing spaces on any given line, and no trailing newline.

Output to screen
```
......_,-ddd888888bbb-._
..:d8888888888888888888b
.:d88888888888888888888888b
:688888888888888888888888889
:68888b8::8q8888888p8::8d88889
.:d8887     p88888q     4888b:
..:d8887    p88888q    4888b:
....:d887   p88888q   488b:
......:d8bod8888888dob8b:
........:d88888888888d:
..........:d8888888b:
............:d8888b:
..............:bd:
```

2. Benford

Program Name: Benford.java Input File: benford.dat

Chris and Matthew are on the committee for a local scientific journal. In the past they've had a problem with accidentally publishing papers whose authors fabricated fraudulent experimental results. As a result, the journal is having serious funding issues and must come up with a way to detect these academically dishonest papers before publishing them.

Luckily for the company, Matt was recently browsing Wikipedia (that's what he does in his free time) and read about a frequency distribution law, called Benford's Law, that states that the digit 1 occurs as a leading digit about 30% of the time in most real life sources of data. Matt wants to write a program that will automatically analyze the distribution of leading digits in any random set of numbers.

Input
The first line of input contains a single integer, *t*, indicating the number of test cases to follow.

The first line of each test case consists of a single integer, *n*, indicating the number of lines in the test case to follow. The next *n* lines each consists of a single integer representing a data point in the test case.

Output
For each test case, print whether the numbers in the test case pass Benford's Law's distribution. A dataset is said to pass Benford's Law if the percentage of numbers that begin with a leading 1 digit is within 30±5%. If the dataset does not pass Benford's Law, print the percentage of numbers that began with a leading 1 digit, rounded to the nearest hundredth.

Constraints
```
1 <= t <= 10
1 <= n <= 10000
```

Example Input File
```
2
4
10
5
9
2
3
17
100
49
```

Example Output to Screen
```
PASSED
FAILED: 66.67%
```

Explanation of sample cases
In the first case, only one of the four numbers begins with a leading 1 digit. That means 25% of the numbers begin with a one. Since this lies between [25%, 35%], the dataset passes. In the second case, two of three numbers begin with a leading 1 digit. This corresponds to a 66.67% one leading numbers, which is outside of the valid range and so the second dataset fails.

3. Bishop

Program Name: Bishop.java Input File: bishop.dat

You're playing chess on a nice *n* x *m* board. Your opponent decided to place *k* bishops around the board. How many positions do you have to safely place your pawn? In chess, a bishop moves and captures along all 4 diagonals leading away from it. It can move as much as the entire length of a diagonal to do so, therefore you are not safe anywhere along any of the diagonals leading away from a bishop.

Input
The first line of input contains *t*, the number of test cases that follow.

For each test case, the first line will consist of three integers, *n*, *m*, and *k*, where *n* represents the number of rows, *m* represents the number of columns, and *k* represents the number of placed bishops. The following *k* lines have two integers each, representing the row and column of a bishop. All positions are 0-indexed.

Output
For each test case, print the number of locations you can safely place your pawn on the board.

Constraints
```
1 <= t <= 10
1 <= n, m <= 1000
0 <= k <= n * m
```

Example Input File
```
3
2 2 1
0 0
2 2 2
0 0
1 0
3 1 2
0 0
1 0
```

Example Output to Screen
```
2
0
1
```

4. Consonant Runs

Program Name: Consonant.java Input File: consonant.dat

Given a string, find the longest run of consonants, case insensitive and which may or may not be consecutive, that are either ascending or descending. In this problem the letter "Y" is a consonant. The following consonants are ascending – "b g t". The following consonants are descending – "m h c". Repeated consonants can be considered ascending when determining ascending runs and descending when considering descending runs. A string of a single consonant can be considered either ascending or descending.

For example, in the string

```
A human must turn information into intelligence or knowledge.
```

the consonants are

```
h m n m s t t r n n f r m t n n t n t l l g n c r k n w l d g
```

Excluding shorter substrings, the ascending runs are

```
hmn, mstt, r, nn, fr, mt, nnt, nt, ll, gn, cr, knw, dg
```

and the descending runs are

```
nm, ttrnnf, rm, tnn, tn, tllg, nc, rk, wld, g
```

The string `ttrnnf` is the longest ascending or descending string, thus the longest run is 6.

Input
The first line of input will consist of a single integer, *n*, indicating the number of lines to follow. Each line will consist of a string between 1 and 128 characters long, inclusive. The string may be a mix of uppercase and lowercase letters, digits, punctuation marks, and spaces.

Output
For each input string, print a single integer on its own line representing the count of the longest run of ascending or descending consonants in the string.

Constraints
```
1 <= n <= 100
```

Example Input File
```
3
A human must turn information into intelligence or knowledge.
$234.56
QWERTY
```

Example Output to Screen
```
6
0
3
```

5. Factorial

Program Name: Factorial.java Input File: factorial.dat

A factorial of a non-negative integer is obtained by multiplying it by each of the non-negative integers leading up to it. For example, 4-factorial, also written as 4!, is equal to 1 x 2 x 3 x 4 = 24. Factorials become very large very quickly. Here are two cases:

```
10! = 3628800
100!  =
933262154439441526816992388562667004907159682643816214685929638952175999932299
156089414639761565182862536979208272237582511852109168640000000000000000000000
00
```

The number of trailing zeroes for 10! is 2. The number of trailing zeroes for 100! is 24. In this problem, you will be given some number *n*, and you will determine the number of trailing zeroes for *n*-factorial.

Input

The first line will consist of a single positive integer *n* that will denote the number of lines of data to follow. The following *n* lines will each consist of a single positive integer m, which will be between 1 and 10,000 inclusive.

Output

You should print the number of trailing zeroes for the factorial of each of the given *n* integers.

Constraints

```
1 <= n <= 10
1 <= m <= 10000
```

Example Input File

```
5
25
103
78
249
34
```

Example Output to Screen

```
6
24
18
59
7
```

6. Helicopter Landing

Program Name: Helicopter.java **Input File: helicopter.dat**

The U.S. Government has recently purchased a large fleet of helicopters. In each city, they need a downtown location to land as many helicopters as possible. However, downtown is already full of buildings. Write a program to help the government find an optimal landing location for each city.

The city can be represented as a two-dimensional grid of numbers, where the number in cell (i, j) is the height of the building at (i, j). Because of how the government's helicopters operate, the landing location must be an axis-aligned rectangle. Furthermore, the heights of all buildings within the rectangle must be the same. The government is interested in the largest such rectangle by area. If multiple potential landing rectangles have the same area, print the one with the greatest height above the city.

Input

The first line of input contains n, the number of cities to examine. For each city, the first line contains 2 integers, w and l, where w is the width of the city's downtown, and l is the length. For the purposes of this problem, let width be distance along the x-axis and length be distance along the y-axis. The next l lines each has w space-separated integers, which are the heights of buildings on a city street.

Output

For each downtown area, give the government a briefing of the landing location, using the following form:

```
Area: a square blocks
Start location: r c
Width: w
Length: l
```

Where a, r, c, w, and l are replaced with the proper values. r is the 0-indexed row number and c is the 0-indexed column number of the top left corner of the solution rectangle. Print a blank line between briefings.

Constraints

```
1 <= n <= 10
1 <= w, l < 100
0 <= (building height) < 10000
```

Example Input File

```
2
3 2
1 3 3
1 3 3
3 3
9 10 9
10 11 10
9 10 9
```

Example Output to Screen

```
Area: 4 square blocks
Start location: 0 1
Width: 2
Length: 2

Area: 1 square blocks
Start location: 1 1
Width: 1
Length: 1
```

7. Mountain

Program Name: Mountain.java Input File: mountain.dat

You love climbing mountains, and in fact, you think of yourself to be quite exceptional at the sport. Your friend decides to test that theory and presents you with a series of challenges. For each challenge, she gives you the layout of a "mountain" of blocks as an *n* x *n* grid of positive integers, where each integer in the grid represents the height of that location in blocks. She tells you your starting position will always be in the top left corner. Given this information, she asks if you think you can reach the top. Rather than rush off to try and prove you really are the greatest mountain climber in town, you decide to approach things logically.

In the grid layout, each block unit is the equivalent of 5 meters. You figure that you can only climb up one single block at a time, but you can jump down up to 2 blocks at a time. Additionally, you can only go up or down adjacent blocks (left, right, up, down).

Armed with this knowledge, you write a program to determine if it is possible for you to reach the highest block.

Input
The first line of input consists of a single integer, *t*, indicating the number of challenges that follow.

For each challenge, there will be a single integer, *n*. The following *n* lines have *n* positive integers each, representing the layout of the "mountain."

Output
For each test case, print `YES` if it's possible to reach the highest point, and `NO` if it's not possible. There will always be a single highest point.

Constraints
```
1 <= t <= 10
1 <= n <= 100
```

Example Input File
```
3
3
1 1 1
1 3 1
1 2 1
3
1 1 1
1 3 1
1 1 2
5
3 1 3 1 1
1 1 3 7 2
2 3 3 7 3
4 4 7 7 4
9 8 7 6 5
```

Example Output to Screen
```
YES
NO
YES
```

8. Rectangles

Program Name: Rectangles.java Input File: rectangles.dat

Given two rectangles and a point in two dimensional space, determine if the point is strictly inside the area of intersection of the two rectangles. If the point is on the perimeter, then it is not strictly inside the intersectional area.

The rectangles are oriented such that their sides are parallel to the x- and y-axes, hence they can be uniquely specified by the coordinates of the upper left corner and the coordinates of the lower right corner. First determine if the two rectangles at all intersect, that is the area of intersection has to be greater than zero. If the two rectangles share a side or a point and the area of intersection is zero, then they do not intersect. If they do intersect, determine the location of the intersection area. Then determine if the given point is inside that intersection area. The two rectangles may partially overlap, one's area may be completely encompassed by the other, or they may be totally disjoint.

Input

The first line in the input file will consist of a single integer *n*, the number of datasets to follow.

Each dataset will be on a line by itself. Each will consist of 10 floating point numbers, separated by one or more spaces, which are arranged as follows: the x- and y-coordinates of the upper left corner and the lower right corner of the first rectangle, the x- and y-coordinates of the upper left corner and lower right corner of the second rectangle, and the x- and y-coordinates of the point in question.

Constraints

`1 <= n <= 10`

Output

For each data set, output `YES` if the given point falls within the intersection area of the two given rectangles. Output `NO` if it does not.

Example Input File

```
2
-3.2 1.5 4.7 -2.8 -2.4 7.3 8.5 -1.6 1.1 0.8
-17.4 9.8 -5.2 3.6 -11.3 7.1 12.4 0.6 -2.5 4.3
```

Example Output to Screen

```
YES
NO
```

9. Spanish Quiz

Program file: SpanishQuiz.java Input File: spanishquiz.dat

You're studying for a quiz in your Spanish class over the countries in Latin America and their capitals. You decide that you could use this opportunity to practice your programming skills and develop a simple flashcard game. Step one is to write the core logic of your program. It must handle quiz questions in two formats, as follows:

```
What is the capital city of <country>?
<City> is the capital city of which country?
```

You have the complete list of countries and corresponding capitals, which follows:

```
Mexico D.F., Mexico
Guatemala, Guatemala
Tegucigalpa, Honduras
San Salvador, El Salvador
Managua, Nicaragua
San Jose, Costa Rica
Panama, Panama
Caracas, Venezuela
Bogota, Colombia
Quito, Ecuador
Lima, Peru
La Paz, Bolivia
Asuncion, Paraguay
Santiago, Chile
Buenos Aires, Argentina
Montevideo, Uruguay
Brasilia, Brazil
```

Input

The first line of input will be a single integer, *n*, indicating the number of quiz questions to follow. Each subsequent line will be a quiz question in one of the two formats given above. You may assume the city or country specified is contained in the above list, and that there are no special characters in any of the names (e.g., Bogota will not have an accent on the 'a').

Constraints

```
1 <= n <= 10
```

Output

Your program must output the correct city or country in response to the input question in the following format:

```
<City> is the capital city of <country>.
```

Example Input File

```
3
What is the capital city of Colombia?
Lima is the capital city of which country?
Tegucigalpa is the capital city of which country?
```

Example Output to Screen

```
Bogota is the capital city of Colombia.
Lima is the capital city of Peru.
Tegucigalpa is the capital city of Honduras.
```

10. Time Zones

Program Name: Timezones.java Input File: timezones.dat

It's interview season and you've managed to land a ton of on-site interviews with different companies. Unfortunately, these companies are situated all over the world and you're having a hard time scheduling the interviews because of all the time differences.

You decide that you need an easier way to convert between these time zones. Why not a program? You quickly put together a list of different time zones and their respective UTC offsets. Your plan is to feed this list, along with the times that you want to convert, into your program and get back a list of converted times.

You can assume that no two time zones will have the same name. Two different time zones could, however, have the same UTC offset.

Input

The first line of input contains two space-separated integers, *n* and *m*, where *n* is the number of time zones and *m* is the number of times to convert.

The next *n* lines contain the *n* different time zones. Each time zone is on its own line and follows this format:

```
CST -06:00
```

where `CST` is replaced with the name of the time zone and `-06:00` is replaced with the respective UTC offset.

The next *m* lines each represents a time to convert. The format for these conversions look like the following:

```
01:20 CST => PST
```

You can assume that times will be given in standard 24-hour military time (00:00 to 23:59).

Output

For each time to convert, print the converted time. Ignore any day changes, and simply print the local time at the destination time zone that corresponds to the local time at the origin time zone.

Constraint

```
1 <= n <= 20
1 <= m <= 100
1 <= length of time zone names <= 16
-12:00 <= UTC offsets <= +14:00
```

Example Input File

```
3 3
CST -06:00
PST -08:00
IST +05:30
12:00 CST => PST
10:00 PST => CST
19:30 PST => IST
```

Example Output to Screen

```
10:00 PST
12:00 CST
09:00 IST
```

11. Typing Test

Program Name: TypingTest.java **Input File: typingtest.dat**

Cole Mak is a slower typist than his friend Dvorak. He wants to improve his typing speed. Help him out by writing a program that measures his speed in words per minute (WPM).

Input

The first line of input contains a single positive integer, *n*, indicating the number of typing tests that Cole takes. Each typing test will be represented by a single line consisting of a number, *t*, followed by one or more space-separated words, where *t* is the number of seconds it took Cole to type out those words.

Consider a word to be a sequence of characters separated by spaces. For example, "dog" and "dog." are both single words (note the period on the end of the second), but "The Hulk" is two words.

Output

For each typing test, print on a single line the number of words per minute Cole typed, rounded to the nearest integer, with units of WPM.

Constraints

```
1 <= n <= 10
1 <= t <= 10000
1 <= (length of a word) <= 15
1 <= (total number of words) <= 10000
```

Example Input File

```
2
9 The quick brown fox jumped over the lazy dog
8 This is the story of a girl who cried a river and drowned the whole world.
```

Example Output to Screen

```
60 WPM
120 WPM
```

12. XML

Program Name: XML.java Input File: xml.dat

Bobby is webmaster for The University of Texas' chapter of the Association for Computing Machinery, or ACM. He stores the ACM members' data in XML format. Files that use XML, or Extensible Markup Language, consist of a hierarchy of tags. Each tag has a name, and possibly a number of children, which are either more tags or plain text. Children are contained within a parent's opening and closing tags, as shown below.

A tag with name "test" and one plain-text child with name "hello" looks like this:

```
<test>hello</test>
```

A tag with name "test" and two children tags with names "t1" and "t2" looks like this:

```
<test><t1>asdf</t1><t2>qwerty</t2></test>
```

Note that t1 and t2 each has one plain-text child.

Bobby recently downloaded a virus that scrambled all of his files and data, changing up the names and mixing around the files. Thankfully, he had an online backup of the files, but he wants to figure out what the virus changed. Bobby wants you to write a program that, given two XML files, says whether the tags are the same. For two tags to be the same, they must have the same name and the same number of children tags with the same names. Child tags may or may not have child tags of their own. Plain text is ignored when making this determination.

Input

The first line of input contains a single positive integer, *n*, indicating the number of pairs of files to process. The next 2*n* lines represent pairs of files, where for each pair, the backup file is on the first line, and the file altered by the virus is on the second line.

Output

For each pair of files, output `same` if the pages have the same tags, or `different` they have different tags.

Constraints

```
1 <= n <= 10
1 <= length of any file <= 100 characters
```

Example Input File

```
4
<name>bobby</name>
<name>bobby</name>
<name>bobby</name>
<name>danny</name>
<name>arby</name>
<lastname>arby</lastname>
<student><name>Katie</name><lastname>Smith</lastname><grade>Junior</grade></st
udent>
<student><name>Katie</name><namelast>Smith</namelast><grade>Junior</grade></st
udent>
```

Example Output to Screen

```
same
same
different
different
```

12. XML (cont.)

Explanation of sample cases

In the first case, both files are exactly the same, so the output is `same`.
In the second case, both tags have the same name, and the plain text is ignored, so the output is `same`.
In the third case, the tag names are different, and the plain text is ignored, so the output is `different`.
In the fourth case, the files have the same outer tag and the same number of children, but one of the children has a different tag name, so the output is `different`.