Taylor High School Programming Contest

Advanced Packet

2010

General instructions:

- You will have two hours to complete the contest.
- There are 18 problems, nine worth 5 points and nine worth 10 points.
- For each 5 point problem, 2 additional points will be awarded to the first team within each division to complete the problem.
- For each 10 point problem, 4 additional points will be awarded to the first team within each division to complete the problem.

Problem 5.1

Signmaking

General Statement
It’s time for another THS contest! Unfortunately, our sign maker forgot to get the signs printed. Your first job is to create a wonderful sign to make sure that people know who we are.

Time Allocation 1 second
Java File Name adv51.java

Data File Name None

Input None

Output See below.

Sample Input None

Sample Output
```
..........................................................
...$$$$$$$$...$.....$$...$$.$$.......$$$$$$$$..$$$$$$$$...
......$$.....$$$....$$...$$.$$.......$$....$$...$$$$$$$...
......$$....$$.$$...$$$$$$$.$$.......$$....$$...$$$.......
......$$...$$$$$$$.......$$.$$.......$$....$$...$$$.......
......$$..$$.....$$......$$.$$.......$$....$$...$$$.......
......$$.$$.......$$.$$$$$$.$$$$$$$..$$$$$$$$...$$$.......
..........................................................
....................$$$$$$$$...$$$$$$$$...................
....................$$.........$$.........................
....................$$.........$$.........................
....................$$.........$$$$$$$$...................
....................$$...............$$...................
....................$$...............$$...................
....................$$$$$$$$...$$$$$$$$...................
..........................................................
```

Problem 5.2

Plentiful Change

General Statement

The day before Halloween, you remember that you were supposed to buy the candy! At the store, you pick up three giant bags of candy. At the counter, you pull out your wallet and notice that you have $50,000 dollars – more than enough to buy the candy, but it is all in twenty dollar bills. Given the individual prices of the bags, output how much you pay and your change.

Time Allocation	1 second
Java File Name	adv52.java
Data File Name	adv52.dat

Input

3 data sets. Each data set includes the three prices of the goods, either integers or doubles, separated by spaces.

Output

Print two dollar values, the amount you pay in cash and your change, each separated by a single space. Output must be formatted exactly like the sample output below.

Assumptions

The maximum sum will be $50,000. Inputs may be doubles or integers. Inputs will not be negative.

Sample Input

```
1.20 22 5
12.65 34.21 7677
1 1 1
```

Sample Output

```
$40.00 $11.80
$7740.00 $16.14
$20.00 $17.00
```

Problem 5.3

Crazy Code

General Statement

You and a friend develop a secret code in order to ensure that certain clandestine activities remain hidden. For each input string, encode the word using the following steps.

1. Convert all characters in the String to upper case.
2. Reverse the word.
3. Use the following cipher. Each top character converts to the one on the bottom, and vice versa.

A	B	C	D	E	F	G	H	I	J	K	L	M
Z	Y	X	W	V	U	T	S	R	Q	P	O	N

4. Replace each letter (not whitespace) with its ASCII value (for ex, ‘A’ would be replaced with “65”).
5. Divide each ASCII value by 10 and print the remainder.

Time Allocation 1 second
Java File Name adv53.java
Data File Name adv53.dat

Input

The first line will consist of a single integer *n*, indicating the number of data sets to come. The next *n* lines will contain a sentence that needs to be encoded.

Output

Output the translated sentences. Each data set should be on a new line.

Assumptions

There will be no punctuation or other non-letters (except whitespace).

Sample Input

```
4
aaabbbcccdddmmmggg
This sentence will look funny encoded
I hope you are having fun programming in this contest
Do not give up no matter how hard it may seem
```

Sample Output

```
444888777888999000
2231 68761762 9928 0669 67705 7676876
2 6563 066 630 472903 705 47288034635 72 2231 1261768
67 167 6924 50 67 361108 863 7303 12 608 8662
```

Problem 5.4

Fun Factors

General Statement

Every number can be expressed as a product of its prime factors.

For example:

$$60 = 2 \times 2 \times 3 \times 5$$

For a given number, print the sum of its prime factors.

NOTE: 1 is not a prime number!

Time Allocation	1 second
Java File Name	adv54.java
Data File Name	adv54.dat

Input

The first line will contain an integer *n* indicating the number of lines to follow. The next *n* lines will contain a single integer *x*.

Output

Print the sum of the prime factors for each integer x.

Assumptions

All inputs are integers. Inputs may be either composite or prime.

$1 < x < 9999999999$

Sample Input

```
2
60
45
```

Sample Output

```
12
11
```

Problem 5.5

Scrambled

General Statement

Determine whether a given word is a scrambled version of another word.

Time Allocation	1 second
Java File Name	adv55.java
Data File Name	adv55.dat

Input

The first line is an integer *n*. The next *n* lines will contain two words, each separated by a single space. Any non-letter and capitalization should be ignored.

Output

If the second word is a scrambled version of the first word, print `yes`. Otherwise output `no`.

Assumptions

Only alphabetic characters should be included when determining whether a word is scrambled. Other characters should be ignored. There will be at least one alphabetic character in each data set.

Sample Input

```
3
Taylor lrytao
High hhgi
School colhoso
```

Sample Output

```
yes
yes
no
```

Problem 5.6

Exciting Equations

General Statement

Each line contains a simple addition equation. Confirm that the equation is correct.

Time Allocation	1 second
Java File Name	adv56.java
Data File Name	adv56.dat

Input

The first line contains an integer n. The next n lines contain an equation that needs to be checked. The equation will be in the form `x+y=z`.

Output

If the equation is correct, print `yes`. Otherwise print `no`.

Assumptions

The equations will be formatted as below. There will be no spaces. All values will be non-negative integers. There will only be a single addition operation.

0 ≤ sum ≤ Integer.MAX_VALUE

Sample Input

```
4
2+3=5
1+1=3
5+2=7
10+9=18
```

Sample Output

```
yes
no
yes
no
```

Problem 5.7

Silly Structures

General Statement

Stacks and queues are two types of structures often used to hold data. Simple queues pop (eject) data in the same order in which they were pushed (inputted). Stacks pop data in the inverse order of input, with the most recently inputted value popped first.

Your job is to define a structure that can be used as both a stack and queue. The structure starts off as a queue that contains, in this order, {'A', 'B', 'C', 'D', 'E', 'F'}. The following methods are defined:

POP	The structure pops and prints the next value from the appropriate side. (In queues, values are removed in the order they are inputted, ie the front. In stacks, values are removed in the reverse order of input, ie the back.)
PUSH x	char x is pushed into the structure.
SWITCH	If the structure is a queue, it becomes a stack. If it is a stack, it becomes a queue.
END	The current data set is finished.

Time Allocation 1 second
Java File Name adv57.java
Data File Name adv57.dat

Input

The first line contains an integer *n*, indicating the number of data sets to come. The next *n* lines will contain a list of the above commands. The last command in each line will be `END`

Output

For each data set, print any string popped from the structure followed by the final structure, separated by a single space. Each data set should be on a new line.

Assumptions

If `POP` is called, the structure will have at least 1 character.

Sample Input

```
1
POP POP PUSH Q PUSH q POP POP SWITCH POP POP PUSH u POP END
```

Sample Output

```
ABCDqQu EF
```

Problem 5.8

Lazy Library

General Statement

The local library system has shut down, so the citizens of Tayloria decide to lend books among each other. However, they need you to develop a computer system to facilitate that system.

Time Allocation	1 second
Java File Name	adv58.java
Data File Name	adv58.dat

Input

The first line of the input will contain an integer n, with the number of data sets. Each data set's first line will contain 2 integers, x and y. The next x lines will be in the form `Name: Book1, Book2, Book3…`. This contains the books that each person in the town owns and has made available for borrowing. The next y lines will contain actions, either returning or borrowing, that a townsperson takes. These lines will either be in the form `Person would like to borrow Book` or `Person returns Book.`

Output

The result of each action. If someone tries to borrow a book that no one owns, print `Sorry, `*`Book`*` does not exist.` If all copies are currently checked out, print *`Sorry`*`, all copies of `*`Book`*` are checked out. `*`Person`*` added to waitlist.` If the book is available, print *`Person`*` borrows `*`Book`*` from `*`Owner`*`.` If multiple copies of the book are available, the Owner with the name first in alphabetical order lends the book.
If returning, print using the form *`Person`*` returns `*`Book`*` to `*`Owner`*`.` If anyone was on the wait-list for that book, print *`Person`*` taken off waitlist and checks out `*`Book`*`.` on the next line – that person now has successfully checked out that book.

Assumptions

No person owns more than one copy of a book. Anyone who returns a book will have borrowed it earlier in the data set. Books and names will only contain capital alphabetic letters, numbers, and whitespace. Each person mentioned in the data set will own at least 1 book.

Sample Input

```
1
4 11
A: B1, B2, B3
C: B2, B3, B4
D: B6, B7, B8
E: B10
A would like to borrow B25
A would like to borrow B4
D would like to borrow B4
E would like to borrow B4
A would like to borrow B10
A returns B10
D would like to borrow B2
E would like to borrow B2
D returns B2
A returns B4
D returns B4
```

Sample Output

```
Sorry, B25 does not exist.
A borrows B4 from C.
Sorry, all copies of B4 are checked out. D added to waitlist.
Sorry, all copies of B4 are checked out. E added to waitlist.
A borrows B10 from E.
A returns B10 to E.
D borrows B2 from A.
E borrows B2 from C.
D returns B2 to A.
A returns B4 to C.
D taken off waitlist and checks out B4.
D returns B4 to C.
E taken off waitlist and checks out B4.
```

Problem 5.9

Additive Addition

General Statement

Input two integers in base 10. Output the sum of the integers in binary, octal, and hex.

Time Allocation 1 second
Java File Name adv59.java
Data File Name adv59.dat

Input

The first line contains integer n. The next n lines are in the form `x y`, where x and y are the two integers to be added.

Output

The sum of the integers in binary, octal, and hex. Output each sum on separate lines. Right justify the output based on the length of the longest string representation. Do not include any leading zeros.

Assumptions

0≤ sum ≤ Integer.MAX_VALUE

Sample Input

```
3
1 2
0 0
111 333
```

Sample Output

```
11
 3
 3
0
0
0
110111100
      674
      1bc
```

Problem 10.1

Dilatory Dependencies

General Statement

Bob landed himself on the Debian Linux repository management team. His new job is to manage incoming packages and ensure that they are properly added to the repository he maintains. A repository is a storage area for software, and a package is a piece of software; ergo repositories hold a group of software. Many packages are dependent on other packages; for instance, the Photo Gallery package requires that libjpg and libpng, which handle JPEG and PNG images respectively, to run. Bob must check packages as they are signed in to the repository to make sure they do not rely on nonexistent packages or create cyclical dependencies (when packages rely on each other).

Thus the two errors that Bob can come across are:

ERRCYCLICAL	Occurs when a package relies on another package that either directly or indirectly relies on it (NOTE: The transitive property does apply). A cyclical error is reported by printing `ERRCYCLICAL` followed by the packages in the cycle in alphabetical order. That is, if x depends on y, y depends on z, and z depends on x, the output should be `ERRCYCLICAL x y z`
ERRBROKENPKGDEP	Occurs if and only if a package **directly** depends on another that does not exist. To report this error, print `ERRBROKENPKGDEP` followed by the package that does not exist and then those packages that depend on that package, in alphabetical order. That is, if x and y depend on z, but z is not in the repository, then the output should be `ERRBROKENPKGDEP z x y`

For instance, Bob may come across data that looks like:

```
pkg photoviewer 3 libpng libjpg libgtk
pkg libpng 0
pkg libjpg 0
pkg libgtk 1 gnome
pkg gnome 0
```

In this repository, `photoviewer` has 3 dependencies: `libpng`, `libjpg`, and `libgtk`. `Libpng` has no dependencies. `Libjpg` has no dependencies. `Libgtk` has 1 dependency: `gnome`. `gnome` has no dependencies. In this situation, the repository is in working condition and should be marked `OK`.

However, if the `libjpg` line was changed to `pkg libjpg 1 photoviewier`, then there would be a cyclical dependency error, because `photoviewer` depends on `libjpg` and `libjpg` depends on `photoviewer`. In this case, an error should be recorded as `ERRCYCLICAL libjpg photoviewer`.

If the gnome package was changed to `pkg gnome 1 libfail`, then there would be a broken package dependency error. This error should be marked as `ERRBROKENPKGDEP libfail gnome`.

Time Allocation	1 second
Java File Name	adv101.java
Data File Name	adv101.dat

Input

The first line will contain an integer, *n*, indicating the number of repositories to be checked for errors. For each repository, the first line consists of two integers, *f* and *p,* indicating the repository's ID number and the number of packages in the repository, respectively. The next *p* lines will each contain strings formatted as `pkg x y <z>`, where `x` is the package identifier, `y` is the number of dependencies the package has, and `<z>` is a list of the dependencies (referred to by package identifier), delimited by spaces.

Output

For each repository, output the line `REPOSITORY F`, where F is the number of the repository. Every error recorded should be printed on a separate line. Each error should be in the format specified in the general statement. Errors should be listed in alphabetical order. If there are no errors, print `NOERRORS`. Print an empty line between each repository.

Assumptions

There will be at least one package in each repository. All package names will be unique and will not contain any whitespace.

Sample Input

```
2
213546 5
pkg photoviewer 3 libpng libjpg libgtk
pkg libpng 0
pkg libjpg 0
pkg libgtk 1 gnome
pkg gnome 0
98464 6
pkg photoviewer 3 libpng libjpg libgtk
pkg hello 2 hi hey
pkg libpng 1 hey
pkg libjpg 0
pkg libgtk 1 gnome
pkg gnome 1 libgtk
```

Sample Output

```
REPOSITORY 213546
NOERRORS

REPOSITORY 98464
ERRBROKENPKGDEP hey hello libpng
ERRBROKENPKGDEP hi hello
ERRCYCLICAL gnome libgtk
```

Problem 10.2

Connect Four

General Statement

You come across a Connect Four board.

In connect four, two players take turns dropping different color pieces from the top into one of ten different columns (the piece will automatically drop to the lowest available spot in a given column). The point of the game is to get four of your pieces in a row, horizontally, vertically, or diagonally. Figure out which player's turn it was from the number of pieces of each color on the board (if there is an equal number of pieces, assume black went first) and then determine whether that player can win on their next turn. Determine the position(s) through which that player can win, or if there is a move with which the player can win at all.

The board is labeled as follows:

A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
A3	B3	C3	D3	E3	F3	G3	H3	I3	J3
A4	B4	C4	D4	E4	F4	G4	H4	I4	J4
A5	B5	C5	D5	E5	F5	G5	H5	I5	J5
A6	B6	C6	D6	E6	F6	G6	H6	I6	J6
A7	B7	C7	D7	E7	F7	G7	H7	I7	J7
A8	B8	C8	D8	E8	F8	G8	H8	I8	J8
A9	B9	C9	D9	E9	F9	G9	H9	I9	J9
A10	B10	C10	D10	E10	F10	G10	H10	I10	J10

Time Allocation 1 second

Java File Name adv102.java

Data File Name adv102.dat

Input

The first line contains integer n, the number of data sets. Each data set consists of two lines. The first contains the initial positions of any black pieces and the second the positions of white pieces.

Output

The color of the player who will play next and the position that they can play to win. If there is none, print `Not Possible`. The possible positions should be displayed from left to right.

Assumptions

The difference between the numbers of pieces of black and white will at most be 1. There will be at least one open spot.

Sample Input

```
2
A10 B10 C10 J10 I9 H8 J9 J8
A9 B9 C9 I10 H10 H9 G10 G9 G8
B10
B9
```

Sample Output

```
Black D10 G7 J7
Not Possible
```

Problem 10.3

Pesky Palindromes

General Statement

Input a large data file with an undisclosed amount of lines filled with palindromes and other words. Print any palindrome within the data set. For our purposes, a palindrome consists of any phrase in which the (alphabetic) letters are the same forward and backward. Punctuation, white spaces, capitalization, numbers, and other non-alphabetic characters are ignored when determining whether a string is a palindrome, but should still be outputted as part of the palindrome. Each palindrome must contain at least 5 letters.

For example, `Don’t nod` will be considered a palindrome, but `(:t;;t:)` is not, because there are not five alphabetic letters in the palindrome.

Time Allocation

2 seconds

Java File Name

pr103.java

Data File Name

pr103.dat

Input

An unspecified number of lines. Treat the entire data file as one large data set. Palindromes can and will be found across multiple lines.

Output

All the palindromes, one on each line, in alphabetic order. Preserve the format, including punctuation, capitalization, numbers, whitespace, and all characters found in the input. An exception will be palindromes that are found across multiple lines. In that case, do not preserve any whitespace at the end of each line.

Assumptions

There will be at least one palindrome.
Palindromes can be across multiple lines.
No two palindromes will contain the same letter sequence.

Sample Input

```
kjnvDon’t nodasdffdsatqwe23ladsfhaioehgqerhg
ghreq
```

Sample Output

```
asdffdsa
Don’t nod
qerhgghreq
```

Problem 10.4

Rectangles

General Statement

Given a set of coordinates, output the perimeter and area of the largest rectangle that can be formed with the given coordinates as corners of the rectangle. (The largest rectangle is the one with the largest area).

Time Allocation	1 seconds
Java File Name	adv104.java
Data File Name	adv104.dat

Input

The first line of the input contains *n*, indicating the number of data sets to follow. The first line of each data set will contain an integer *t*. The next *t* lines include two integers x and y, which indicate the x and y coordinates of each point. x and y will be separated by a single space.

Output

For each data set output print the perimeter and area of the largest rectangle that can be formed with the given coordinates, each rounded to 2 decimal places. Print an empty space between the perimeter and area. Each data set should be on a new line.

Assumptions

All coordinates will be non-negative integers. At least one rectangle will be able to be formed.

Sample Input

```
2
4
0 0
2 2
0 2
2 0
5
0 0
5 1
9 2
4 5
8 6
```

Sample Output

```
8.00 4.00
16.49 17.00
```

Problem 10.5

I would suggest you always read the packet all the way through.

General Statement

Welcome to the easiest problem in the packet.

Time Allocation	1 seconds
Java File Name	adv105.java
Data File Name	None
Input	None

Output

Print the following line, just once.

```
I will always read the packet all the way through.
```

Assumptions

There is no trick to this problem. It is an easy 10 points (or 14, if you are fast enough).

Sample Input

None

Sample Output

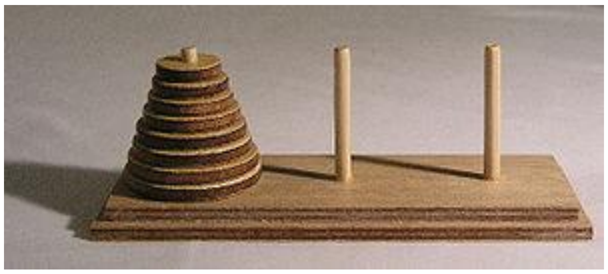
```
I will always read the packet all the way through.
```

Problem 10.6

Towers of Hanoi

General Statement

The Towers of Hanoi, a puzzle game first invented in 1883, is believed by some to predict the end of the world. According to legend, Vietnamese priests are slowly progressing in solving the puzzle with 64 disks, and when they are finished, the world will end. Don't worry: even if they work at the rate of one move per second, it will take almost 600 Billion years to complete! Your job is to print the steps to complete the game, albeit with less disks.



The goal of the game is to transfer all the disks from the first tower to the last one. You can only move one disk at the time, and a disk can only be in an otherwise empty tower or on top of a bigger disk. For our game, the smallest disk will be '1' and the largest disk will be x – the number of disks present. The towers are labeled, from left to right, 'A', 'B', 'C'.

NOTE: The least number of moves it takes to solve a game with *x* disks is 2^x-1.

Time Allocation	1 seconds
Java File Name	adv106.java
Data File Name	adv106.dat

Input

The first line will contain an integer n, the number of data sets. The next n lines contain a single integer that gives the number of disks in the current game.

Output

The order of moves it takes to solve the tower most efficiently. For each data set, print `Data set x:` on the first line, where x is the number of the data set. Then print the moves in order that it takes to solve the game most efficiently. This should be in the format `Move disk y to tower z`, where y is the disk number and z is the tower name. The last line of each data set output should be the number of moves it took to complete the game, in the format `Game solved in w move(s).` Print an empty line between each data set.

Assumptions

1 ≤ disks ≤ 9

Sample Input
```
3
1
3
2
```
Sample Output
```
Data set 1:
Move disk 1 to tower C
Game solved in 1 move(s).

Data set 2:
Move disk 1 to tower C
Move disk 2 to tower B
Move disk 1 to tower B
Move disk 3 to tower C
Move disk 1 to tower A
Move disk 2 to tower C
Move disk 1 to tower C
Game solved in 7 move(s).

Data set 3:
Move disk 1 to tower B
Move disk 2 to tower C
Move disk 1 to tower C
Game solved in 3 move(s).
```

Problem 10.7

Walk in the Park

General Statement

Multiple people are in a wide-open field walking in various directions at various rates. Write a program determining the closest distance each person gets to every other person at any single point of time.

Time Allocation	1 seconds
Java File Name	adv107.java
Data File Name	adv107.dat

Input

The first line contains integer n, representing the number of data sets. The first line of each data set includes integer p, representing the number of people in the field. The next p lines will contain a string, j, followed by 4 integers: `x`, `y`, `d`, and `r`. `j` refers to the name of the person. `x` and `y` represent the starting coordinates, `d` represents the degree angle of the direction of travel with respect to the positive x axis. `r` represents the person's speed, in units per second.

Output

Print the list of closest distances between each pair, from least to greatest, rounded to the nearest integer. Each line of the list should be in the form "Name1 Name2 distance"; where the Name1 and Name2 indicate the two people whose distances were measured, in alphabetical order and distance indicates the closest distance between the two. If two different pairs have the same closest distance (rounded to the nearest integer), order them alphabetically by Name1; if the two pairs have the same Name1, order them alphabetically by Name2.

Assumptions

Starting coordinates will be positive integers.

0º ≤ d ≤ 360º

Time should be increased by .1 seconds. (ie, calculate the distance between each pair every .1 seconds, starting at time = 0)

Sample Input

```
1
4
Bob 0 0 45 1
Kate 2 0 135 1
Lion 1 1 179 0
Liger 4 5 160 3
```

Sample Output

```
Bob Kate 0
Bob Lion 0
Kate Lion 0
Bob Liger 5
Kate Liger 5
Liger Lion 5
```

Problem 10.8

You will be baked. And then there will be cake.

General Statement

One Halloween day, Bob decided that instead of going anywhere fun, he would attend a computer science contest. But, before he went, he had to make sure that he was completely prepared. He made a list of everything he needed: a computer, a backup computer, a printer, a backup printer, 3 blue pens, 2 black pens, 5 number two pencils, 1 number three pencil, 4 flashdrives, 2 compact disks, 8 floppy disks, a mousepad, a backup mousepad, an external keyboard, a backup external keyboard, a mouse, a backup mouse, a monitor, a backup monitor, backups to the backups, Ethernet cords, his physics textbook (for when he is bored before the contest), a permission slip, and $10 for pizza. Before he leaves to buy all the materials, he needs to make sure that he has enough money – however, before his mother gives him that money, he needs to do all of his chores: mow the lawn, vacuum, clean his room, recalculate gravity, build a car, clean that car, change its oil, write a new linux distro, wash the cat, beat 3D pinball for windows, clean all the trophies he got from previous computer science contests, wax those trophies, clean them again, clean his room again (which has gotten dirty in the process of doing all his other chores), and then finally go to the gas station and buy gasoline. On his way to the gas station, he stops and notices that his brakes don't work! Unfortunately, Bob was speeding at 265 km/h on the autobahn when he found out (he is in Germany by the way) and lost control of his car. Luckily, Bob, being the prepared computer scientist that he is, manages to stop his car in a way that will not be described. However, a police officer saw how he stopped, and transported him to the Aperture Science Enrichment Center, where he is to be a test subject. Bob must make his way to the exit of the center as fast as possible and escape, in the least amount of steps. However, to leave the building, he must bribe GlaDOS with cake and so must pick that up on the way. Also, some walls are lined with turrets that will shoot and kill Bob if he steps in its line of vision.

Time Allocation	1 seconds
Java File Name	adv108.java
Data File Name	adv108.dat

Input

The first line of input will be an integer *x* that represents the number of data collections to follow. The first line of each data collection will contain two integers *r* and *c,* separated by a single space, that represents the number of rows and columns, respectively, in the maze. The next *r* lines will contain the maze. Use the following key for the maze:

.	Empty space – Bob can walk on these unless otherwise specified.
S	Start
C	The cake. Bob must visit this point before heading to the finish.
F	Finish.
>	A turret facing the right. Bob cannot walk on this or any continuous open spaces to the right (stopping at the next wall)
<	A turret facing the left. Bob cannot walk on this or any continuous open spaces to the left (stopping at the next wall)
*	Wall. Bob cannot cross these.

Output
The minimum number of steps it takes for Bob to grab the cake and head to the finish.

Assumptions
There is at least one possible route from start to finish. Bob can step on the same square twice if necessary.

Sample Input
```
2
10 10
S*********
...*******
**.*******
.........C
..**..**..
.***.*.**.
..........
..........
*********.
F.........
10 10
S*********
...*******
**.*******
...>......
..**..**..
.***.*.**.
..........
....*...<*
*.***.***.
F........C
```

Sample Output
```
27
31
```

Problem 10.9

Wonderful Word Squares.

General Statement

For an Integer x, place the first x letters of the alphabet into a square matrix, starting at the bottom left corner of the square and spiraling in a counterclockwise direction. If x is not a perfect square or if x is greater than 26, use an asterisk for each extra space in the matrix.

Note: The matrix must have the same number of rows as it does columns. Use the smallest matrix size possible to contain the given number of elements. For ex, If x is 17, use a 5 by 5 matrix, with the first 17 letters of the alphabet followed by 8 asterisks.

Time Allocation	1 seconds
Java File Name	adv109.java
Data File Name	adv109.dat

Input

The first line of input will be an integer *n* that represents the number of data collections to follow. The next *n* lines will each contain an integer x, representing x as described above.

Output

Output the square generated. Print an empty line between datasets.

Assumptions

All inputs will be positive integers.

Sample Input

```
3
8
16
27
```

Sample Output

```
GFE
H*D
ABC

JIHG
KPOF
LMNE
ABCD

PONMLK
Q****J
R***ZI
S***YH
TUVWXG
ABCDEF
```