1. Candies for Kids

Program Name: Candy.java Input File: candy.dat

Patty is a kindergarten teacher in Texas. She likes to reward her students for good behavior by giving them packages of candy. She has several sets of packages of candy but, for any given set, there is not the same number of candies in each package. Fearing that a student would complain about someone else receiving more candies than a friend received, she has decided to open all of the packages in each set and move some candies so each package contains the same number of candies.

You are to determine how many candies Patty has to move into or out of each package in a set to ensure that each package contains the same number of candies. If there are extra candies after each package is complete, Patty will make a package for herself using the left-over candies. All candies in each set must be used and the number of candies that Patty will have in her package will always be less than the number of candies in a student package.

For example, Patty has a set of three packages that contain 3, 4, and 6 candies respectively. She needs to put one candy in package 1, zero candies in package 2, and take 2 candies out of package 3. She will have one candy left for her package. Therefore, the output would be: `1 0 -2 1`

Input
The first line will contain a single integer `n` that indicates the number of sets of packages that will follow. Each of the next `n` lines will have `p` integers that indicate the number of candies in the packages for that set.

Output
For each package in a set, you are to print the number of candies that need to be moved into or removed from the package (a negative value) so all packages have the same number of candies, followed by the number of candies that Patty will have in her package. Results for each set will be on a single line and be separated by a single space.

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

Example Output to Screen
```
1 1 1 1 -4 0
2 1 0 -1 -2 -3 3
-1 1 2 -2 0 0
```

2. Code Two

Program Name: Code2.java Input File: code2.dat

Sally and Nancy have decided to write to each other in a special code. They decide to use the smallest square matrix that will hold all of the characters in their message. They will place their message, one character per cell, into the matrix in upside-down column-row order and fill any remaining matrix cells with an asterisk (*). For example, the coded message:

```
eaetoo**hrmedc**todmir.*sfooaiy*ieoceer*wmgohht*oiltttn*Ntlnofu*
```

would have been placed in an 8x8 matrix as:

```
eaetoo**
hrmedc**
todmir.*
sfooaiy*
ieoceer*
wmgohht*
oiltttn*
Ntlnofu*
```

so it could be decoded by reading column one beginning at the bottom and going up, then column two, and so forth and finally getting the message:

```
Nowisthetimeforallgoodmentocometotheaidoftheircountry.**********
```

Input

The first line of input will contain a single integer n that indicates the number of coded messages to follow. Each of the following n lines will contain a single coded message. Each message will contain a perfect square number of characters. There will be no spaces in the input.

Output

For each line of input, you will print the decoded message with no spaces and with the filler asterisks at the end.

Example Input File

```
3
eaetoo**hrmedc**todmir.*sfooaiy*ieoceer*wmgohht*oiltttn*Ntlnofu*
acicisynmsadlaeelalNrloeadadohSnydhr
Se0*Lt2*Ian9Uti0
```

Example Output to Screen

```
Nowisthetimeforallgoodmentocometotheaidoftheircountry.**********
SallyandNancyaremiddleschoolairheads
UILStatein2009**
```

3. Lightning

Program Name: Lightning.java Input File: none

David wants to create a lightning symbol for his band's logo. You are to write a program that will print this design for him.

Input
There is no input for this problem.

Output
Print this lightning design exactly as it appears below.

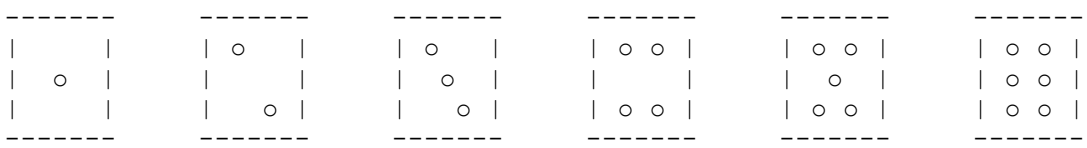
Example Input File
There is no input for this problem.

Example Output to Screen
```
*****************************
*    ************************
****    ***********************
*******    ********************
**********    *****************
*************    *************
****************    **********
******************    *******
**********************    ****
*************************    *
**********************    ****
******************    *******
****************    **********
*************    *************
**********    *****************
*******    ********************
****    ***********************
*    ************************
****    ***********************
*******    ********************
**********    *****************
*************    *************
****************    **********
******************    *******
**********************    ****
*************************    *
*****************************
```

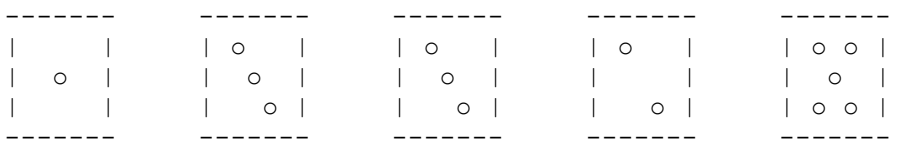
4. Petals Around the Rose

Program Name: Petals.java Input File: petals.dat

Petals Around the Rose is a game that uses 5 six-sided dice like these (7 characters across and 5 characters down) with each dot on the die represented by a lower case 'o'. Only one side of each die is used for this game and each die used will be represented by one of the following 5 x 7 character matrices:



For each game, the five dice are rolled. Perhaps this is the result of the roll for a game:



The object of the game is for the players who do not know how to count the petals to figure out how to count the petals. For the purpose of this program, I will tell you how to count the petals. If there is a dot in the middle of the die, then count the "petals" around the dot. In this above example, the one, both the threes and the five have a dot in the middle. There are no "petals" around the middle dot on the one, there are two "petals" around the middle dot on each of the threes, and there are 4 "petals" around the middle dot on the five. The two has no middle dot so it has no "petals". Therefore, in this game, there are 8 Petals Around the Rose.

Input

The first line of input will contain a single integer `n` indicating the number of games that will be played. For each game, there will be five sets of five lines with each set representing a single die (7 characters across x 5 characters down).

Output

For each game, you will print the number of Petals Around the Rose.

Example Input File

```
2
-------
|     |
|  o  |
|     |
-------
-------
| o   |
|  o  |
|   o |
-------
-------
| o   |
|  o  |
|   o |
-------
-------
| o   |
|     |
|   o |
-------
-------
| o o |
|  o  |
| o o |
-------
-------
| o   |
|  o  |
|   o |
-------
-------
| o   |
|     |
|   o |
-------
-------
| o o |
| o o |
| o o |
-------
-------
| o o |
|     |
| o o |
-------
-------
| o o |
|  o  |
| o o |
-------
```

Example Output to Screen

```
8
6
```

5. The Thief's Dilemma

Program Name: Thief.java Input File: thief.dat

Danny the thief has hit the jackpot. He has successfully broken into a museum filled with valuable items. The problem is there are more items he wants to take than he can possibly carry. Danny has an upper weight limit he can carry. Each item he wants to take has a weight and a value. Write a program that prints out the highest possible total value of items Danny can take, given various weight limits Danny can carry and various lists of items.

Consider this example. Danny's weight limit is 15. There are three items to choose from. Item 1 has a weight of 9 and a value of 12. Item 2 has a weight of 7 and a value of 7. Item 3 has a weight of 7 and a value of 6. The total weight of the items Danny selects must be less than or equal to the weight he can carry while maximizing the value of the items selected. The solution that provides the maximum value is to take item 2 and 3. The total weight is 14 and the total value is 13. If he had chosen item 1 with a weight of 9 he could not have carried either of the other items and the value would have only been 12.

Input

- The first line will contain a single integer `n` that indicates the number of data sets that follow.
- The first line in each data set will be an integer `c` that indicates the maximum weight Danny can carry for this data set. The maximum weight `c` will be greater than 0 and less than 100.
- The second line in each data set will be an integer `m` that indicates the number of items in this data set. The number of items `m` will be greater than 0 and less than 20.
- The third line in each data set will contain `m` pairs of the form `[w,v]`.
    - `w` represents the weight of that item. All weights will be greater than 0 and less than 100.
    - `v` represents the value of that item. All values will be greater than 0 and less than 1000.
    - Each pair will be separated by a single space.

Output

For each data set print out `DATA SET <#>` where `<#>` is the number of the data set starting at 1, followed by the highest possible value of items Danny can take given the weight limit and the items in the data set. The sum of the weights of items chosen must be less than or equal to the maximum weight Danny can carry.

Example Input File

```
3
15
5
[12,4] [2,2] [2,1] [1,1] [4,10]
15
3
[9,12] [7,7] [7,6]
20
6
[5,5] [6,3] [12,4] [3,2] [10,12] [7,4]
```

Example Output To Screen

```
DATA SET 1 14
DATA SET 2 13
DATA SET 3 19
```

6. Transformers: Optimus Prime

Program Name: Transform.java **Input File: transform.dat**

Sam wants to send a message to his friend Mikaela but is afraid of the message being read by others. Sam has a simple encryption scheme to apply to his message. Instead of sending characters Sam will send numbers. He wants to use the ASCII values of characters but that is a little too obvious. Everyone knows 'A' is 65, right? So he will represent the numbers using bases other than base ten. To make his code harder to break he is going to change the base for each line. The first line of the message will use base two, the second line will use base three, the third line will use base five, the fourth line will use base seven. If there are more than four lines in the message the fifth line will cycle back to base two, the sixth line will be base three and so forth. Here is an example of which base to use on a given line

`Line 1` -- uses base two
`Line 2` -- uses base three
`Line 3` -- uses base five
`Line 4` -- bases seven
`Line 5` -- uses base two
`Line 6` -- uses base three
`Line 7` -- uses base five
`Line 8` -- uses base seven
`Line 9` -- uses base two
...

Each character in the message will be replaced by a number, including spaces. The number will be the ASCII value of the character in the appropriate base. The newline characters at the end of the line are not encoded. The number for each character will be followed by a single underscore character, `'_'`, except for the last character on a given line.

Input
The first line of input will contain a single integer `n` that indicates the number lines in Sam's message. The following `n` lines will be the plain text version of Sam's message.

Output
Output the encoded messaging using Sam's encryption algorithm.

Example Input File
```
5
Mik
Hi!
How are u?
"car" is yellow
Bye :)
```

Example Output To Screen
```
1001101_1101001_1101011
2200_10220_1020_1012
242_421_434_112_342_424_401_112_432_223
46_201_166_222_46_44_210_223_44_232_203_213_213_216_230
1000010_1111001_1100101_100000_111010_101001
```