CS 11 Exercise 07
1st Semester, AY 2018-2019

University of the Philippines, Diliman

October 13, 2018

Instructions

- For this exercise, you have to write five different computer programs.
- Make sure that your algorithm works given the sample input and output. You must also check if your algorithm can also handle input other than the ones given.
- Please remove any prompt messages (e.g. `Enter number:`) when getting input. Prompt messages will mess with your output, making your solution invalid.
- See the sample input and output to guide you on what and how your program must display output.
- Submit your solutions via HackerRank. The HackerRank site is posted in UVLe.
- Submit your solutions on or before Sunday, October 28 at 11:59pm.

1 Having Fun with Matrices

We have a matrix of size $N \times N$. Each value of the matrix occupies an integer from $[0, 9]$. A few operations are going to be performed on this matrix. We would like to know how the matrix looks like after these operations are performed sequentially.

There could be five different types of operations.

- `row` a b
  In this operation, row a is interchanged with row b. Rows are labeled from 1 to N, with 1 being the top row and N being the bottom row.
- `col` a b
  In this operation, column a is interchanged with column b. Columns are labeled from 1 to N, with 1 being the leftmost column and N being the rightmost column.
- `inc`
  In this operation, every cell value is increased by 1 (modulo 10). That is, if after adding 1, a cell value becomes 10 we change it to 0.
- `dec`
  In this operation, every cell value is decreased by 1 (modulo 10). That is, if after subtracting 1, a cell value becomes -1 we change it to 9.
- `transpose`
  In this operation, we simply transpose the matrix. Transposing a matrix A, denoted by A^T, means turning all the rows of the given matrix into columns and vice-versa.

  Example:

```
1 2 3                                 1 4 7
4 5 6     after transposing becomes   2 5 8
7 8 9                                 3 6 9
```

1.1 Input

The input starts with an integer T ($T < 50$) that indicates the number of test cases. Each case starts with a positive integer N ($N < 10$) that represents the size of the matrix. The next N lines contain N integers each. The value of each integer is in the range $[0, 9]$. Next there is a line with an integer M ($M < 50$). Each of the next M lines contain an operation each. If the command is '`row` a b' or '`col` a b', then you can assume $1 \leq a, b \leq N$ and $a \neq b$.

You can find this problem at `https://uva.onlinejudge.org` with problem ID 11360.

1.1.1 Sample Input

```
2
4
1234
5678
1234
5678
1
transpose
3
000
111
000
2
row 1 2
inc
```

1.2 Output

For each case, output the case number on the first line. Then on the next N lines output the content of the final matrix. Print a blank line after each case (even after the very last one).

1.2.1 Sample Output

```
Case #1
1515
2626
3737
4848

Case #2
222
111
111
```

2 LC-Display

A friend of yours has just bought a new computer. Before this, the most powerful machine he ever used was a pocket calculator. He is a little disappointed because he liked the LCD display of his calculator more than the screen on his new computer! To make him happy, write a program that prints numbers in LCD display style.

2.1 Input

The input file contains several lines, one for each number to be displayed. Each line contains integers s and n, where n is the number to be displayed ($0 \leq n \leq 99,999,999$) and s is the size in which it shall be displayed ($1 \leq s \leq 10$). The input will be terminated by a line containing two zeros, which should not be processed.

2.1.1 Sample Input

```
2 12345
3 67890
0 0
```

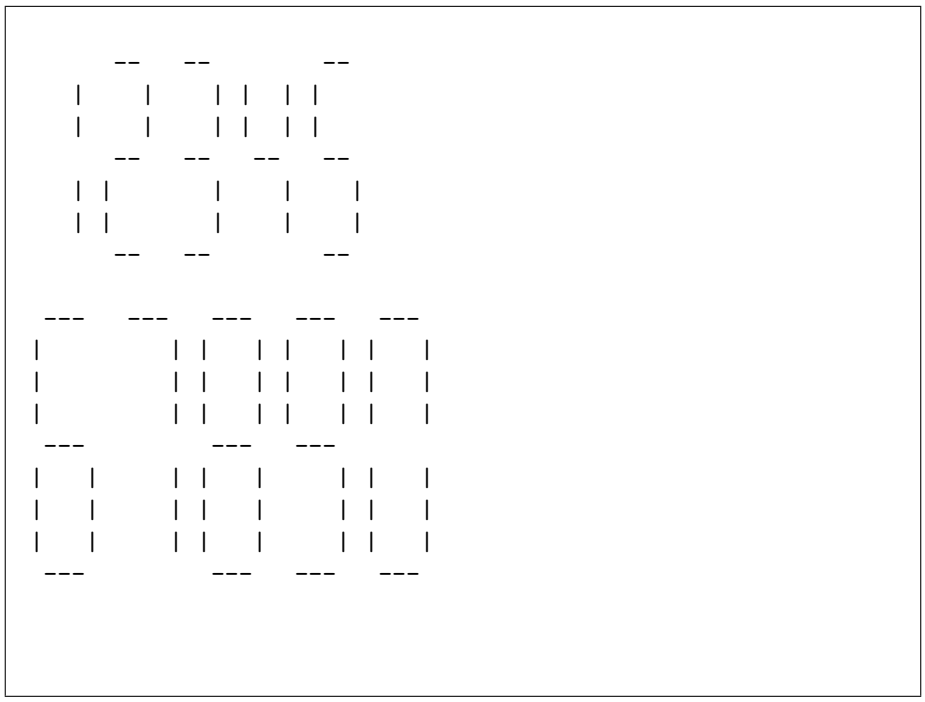
This problem is borrowed from the book *Programming Challenges: The Programming Contest Training Manual* by Steven Skiena and Miguel Revilla. You can find this problem at `https://uva.onlinejudge.org` with problem ID 706.

2.2 Output

Print the numbers specified in the input file in an LCD display-style using s "-" signs for the horizontal segments and s "—" signs for the vertical ones. Each digit occupies exactly $s+2$ columns and $2s+3$ rows. Be sure to fill all the white space occupied by the digits with blanks, including the last digit. There must be exactly one column of blanks between two digits.

Output a blank line after each number. You will find an example of each digit in the sample output below.

2.2.1 Sample Output



3 Where's Waldorf?

Given a m by n grid of letters, ($1 \leq m, n \leq 50$), and a list of words, find the location in the grid at which the word can be found.

A word matches a straight, uninterrupted line of letters in the grid. A word can match the letters in the grid regardless of case (i.e. upper and lower case letters are to be treated as the same). The matching can be done in any of the eight directions either horizontally, vertically or diagonally through the grid.

3.1 Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The input begins with a pair of integers, m followed by n, $1 \leq m, n \leq 50$ in decimal notation on a single line. The next m lines contain n letters each; this is the grid of letters in which the words of the list must be found. The letters in the grid may be in upper or lower case. Following the grid of letters, another integer k appears on a line by itself ($1 \leq k \leq 20$). The next k lines of input contain the list of words to search for, one word per line. These words may contain upper and lower case letters only (no spaces, hyphens or other non-alphabetic characters).

This problem is borrowed from the book *Programming Challenges: The Programming Contest Training Manual* by Steven Skiena and Miguel Revilla. You can find this problem at `https://uva.onlinejudge.org` with problem ID 10010.

3.1.1 Sample Input

```
2

8 11
abcDEFGhigg
hEbkWalDork
FtyAwaldORm
FtsimrLqsrc
byoArBeDeyv
Klcbqwikomk
strEBGadhrb
yUiqlxcnBjf
4
Waldorf
Bambi
Betty
Dagbert

9 8
jajPjpVd
WJVhyowL
YZMZrqDG
rxpMiqCo
RKGCzlmG
KRjRKZKR
DIMJfQmy
dcAhazYQ
uOLWukMH
4
Q
QaW
m
au
```

3.2 Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

For each word in the word list, a pair of integers representing the location of the corresponding word in the grid must be output. The integers must be separated by a single space. The first integer is the line in the grid where the first letter of the given word can be found (1 represents the topmost line in the grid, and m represents the bottommost line). The second integer is the column in the grid where the first letter of the given word can be found (1 represents the leftmost column in the grid, and n represents the rightmost column in the grid). If a word can be found more than once in the grid, then the location which is output should correspond to the uppermost occurence of the word (i.e. the occurence which places the first letter of the word closest to the top of the grid). If two or more words are uppermost, the output should correspond to the leftmost of these occurences. All words can be found at least once in the grid.

3.2.1 Sample Output

```
2 5
2 3
1 2
7 8

3 6
7 6
3 3
8 5
```

4 Minesweeper

Have you ever played Minesweeper? It's a cute little game which comes within a certain Operating System which name we can't really remember. Well, the goal of the game is to find where are all the mines within an $M \times N$ field. To help you, the game shows a number in a square which tells you how many mines there are adjacent to that square. For instance, suppose the following 4×4 field with 2 mines (which are represented by an '*' character):

```
*...
....
.*..
....
```

If we would represent the same field placing the hint numbers described above, we would end up with:

```
*100
2210
1*10
1110
```

As you may have already noticed, each square may have at most 8 adjacent squares.

4.1 Input

The input will consist of an arbitrary number of fields. The first line of each field contains two integers n and m ($0 < n, m \leq 100$) which stands for the number of lines and columns of the field respectively. The next n lines contains exactly m characters and represent the field.

Each safe square is represented by an '.'. character (without the quotes) and each mine square is represented by an '*' character (also without the quotes). The first field line where $n = m = 0$ represents the end of input and should not be processed.

This problem is borrowed from the book *Programming Challenges: The Programming Contest Training Manual* by Steven Skiena and Miguel Revilla. You can find this problem at `https://uva.onlinejudge.org` with problem ID 10189.

4.2 Sample Input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

4.3 Output

For each field, you must print the following message in a line alone:

`Field #`x`:`

Where x stands for the number of the field (starting from 1). The next n lines should contain the field with the '.' characters replaced by the number of adjacent mines to that square. There must be an empty line between field outputs.

4.3.1 Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

5 Check The Check

Your task is to write a program that reads a chess board configuration and answers if there's a king under attack (i.e. "in check"). A king is in check if it's in a square which is attacked by an opponent's piece (i.e. it's in square which can be taken by an opponent's piece in his next move).

White pieces will be represented by uppercase letters whereas black pieces will be represented by lowercase letters. White side will always be on the bottom of the board and black side will always be on the top of the board.

For those unfamiliar with chess, here are the movements of each piece:

Pawn (p or P): can only move straight ahead, one square at a time. But it takes pieces diagonally (and that's what concerns to you in this problem).

Knight (n or N): have a special movement and it's the only piece that can jump over other pieces. The knight movement can be viewed as an "L". See the example below.

Bishop (b or B): can move any number of squares diagonally (forward or backward).

Rook (r or R): can move any number of squares vertically or horizontally (forward or backward).

Queen (q or Q): can move any number of squares in any direction (diagonally, horizontally or vertically, forward or backward).

King (k or K): can move one square at a time, in any direction (diagonally, horizontally or vertically, forward or backward).

Movements examples ('*' indicates where the piece can take another pieces):

```
Pawn       Rook       Bishop     Queen      King       Knight
........   ...*....   .......*   ...*...*   ........   ........
........   ...*....   *.....*.   *..*..*.   ........   ........
........   ...*....   .*...*..   .*.*.*..   ........   ..*.*...
........   ...*....   ..*.*...   ..***...   ..***...   .*...*..
...p....   ***r****   ...b....   ***q****   ..*k*...   ...n....
..*.*...   ...*....   ..*.*...   ..***...   ..***...   .*...*..
........   ...*....   .*...*..   .*.*.*..   ........   ..*.*...
........   ...*....   *.....*.   *..*..*.   ........   ........
```

This problem is borrowed from the book *Programming Challenges: The Programming Contest Training Manual* by Steven Skiena and Miguel Revilla. You can find this problem at `https://uva.onlinejudge.org` with problem ID 10196.

Remember that the knight is the only piece that can jump over other pieces. The pawn movement will depend on its side. If it's a black pawn, it can only move one square diagonally down the board. If it's a white pawn, it can only move one square diagonally up the board. The example above is a black pawn as it's a lowercase 'p' (we say "move" meaning the squares where the pawn can move to when it takes another piece)

5.1 Input

There will be an arbitrary number of board con gurations on the input. Each board will consist of 8 lines of 8 characters each. A '.' character will represent an empty square. Upper and lower case letters (as defined above) will represent the pieces. There will be no invalid characters (i.e. pieces) and there won't be a configuration where both kings are in check. You must read until you find an empty board (i.e. a board that is formed only of '.' characters) which should not be processed. There will be an empty line between each pair of board configurations. In all boards (except the last one which is empty) will appear both the white king and the black king (one, and only one of each).

5.1.1 Sample Input

```
..k.....
ppp.pppp
........
.R...B..
........
........
PPPPPPPP
K.......

rnbqkbnr
pppppppp
........
........
........
........
PPPPPPPP
RNBQKBNR

rnbqk.nr
ppp..ppp
....p...
...p....
.bPP....
.....N..
PP..PPPP
RNBQKB.R

........
........
........
........
........
........
........
........
```

5.2 Output

For each board configuration read you must output one of the following answers:

```
Game #d:  white king is in check.
Game #d:  black king is in check.
Game #d:  no king is in check.
```

Where d stands for the game number (starting from 1).

5.2.1 Sample Output

```
Game #1: black king is in check.
Game #2: no king is in check.
Game #3: white king is in check.
```