

TD n°1

1 Useful information

1.1 Problems

Eight or more problems have been posed in recent World Finals.

Problems will be posed in English. During the contest, all communications from contest officials to contestants will be in English. Each team may identify an interpreter for translating questions posed by contestants to contest officials. Contestants may bring electronic natural language translators provided that they do not support math operations. Each team will be permitted to provide a PDF file of up to 25 pages of notes.

Solutions to problems submitted for judging are called runs. Each run is judged as accepted or rejected, and the team is notified of the results. Rejected runs will be marked as follows:

- run-time error
- time-limit exceeded
- wrong answer
- presentation error

1.2 Provided Computer

- Software
 - OS: CentOS 5.2 Linux
 - Desktop: KDE & GNOME
 - Editors: vi/vim, gvim, emacs, gedit, kate
 - Languages: Java (version 1.6.0_07), C/C++ (GCC 4.1.2 20071124)
 - IDE: Java - Eclipse 3.4.1, C/C++ - CDT 5.0.1 under Eclipse 3.4.1
 - PC² Contest Control System, Version 9.1
- Reference Materials: These packages will be installed automatically as part of the installation instructions that will be listed under detailed installation.
 - JDK JavaDocs
 - C++ STL docs

Keyboard will most certainly have an American layout.

1.3 A few links

- Official webpage: <http://cm2prod.baylor.edu/login.jsf>
- UVa Online Judge: <http://icpcres.ecs.baylor.edu/onlinejudge/>. **You need to create an account on this website to submit your codes.**
- C/C++:
 - A nice introduction to C (in French): http://www-clips.imag.fr/commun/bernard.cassagne/Introduction_ANSI_C.html
 - STL webpage: <http://www.sgi.com/tech/stl/>

- Java:

- A tutorial: <http://erci.no-ip.com/origine/divers/java/tutorial-java/index.html>
- Java SE 6 API: <http://java.sun.com/javase/6/docs/api/>

2 Minesweeper (10189)

2.1 The Problem

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 a $M \times N$ 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.

2.2 The 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.

2.3 The 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.

2.4 Sample Input

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

2.5 Sample Output

```
Field #1:
*100
```

2210
1*10
1110

Field #2:

**100
33200
1*100

3 The Trip (10137)

A number of students are members of a club that travels annually to exotic locations. Their destinations in the past have included Indianapolis, Phoenix, Nashville, Philadelphia, San Jose, and Atlanta. This spring they are planning a trip to Eindhoven.

The group agrees in advance to share expenses equally, but it is not practical to have them share every expense as it occurs. So individuals in the group pay for particular things, like meals, hotels, taxi rides, plane tickets, etc. After the trip, each student's expenses are tallied and money is exchanged so that the net cost to each is the same, to within one cent. In the past, this money exchange has been tedious and time consuming. Your job is to compute, from a list of expenses, the minimum amount of money that must change hands in order to equalize (within a cent) all the students' costs.

3.1 The Input

Standard input will contain the information for several trips. The information for each trip consists of a line containing a positive integer, n , the number of students on the trip, followed by n lines of input, each containing the amount, in dollars and cents, spent by a student. There are no more than 1000 students and no student spent more than \$10,000.00. A single line containing 0 follows the information for the last trip.

3.2 The Output

For each trip, output a line stating the total amount of money, in dollars and cents, that must be exchanged to equalize the students' costs.

3.3 Sample Input

3
10.00
20.00
30.00
4
15.00
15.01
3.00
3.01
0

3.4 Output for Sample Input

\$10.00
\$11.99

4 LC Display (00706)

A friend of you has just bought a new computer. Until now, the most powerful computer he ever used has been a pocket calculator. Now, looking at his new computer, he is a bit disappointed, because he liked the LC-display of his calculator so much. So you decide to write a program that displays numbers in an LC-display-like style on his computer.

4.1 Input

The input file contains several lines, one for each number to be displayed. Each line contains two integers s, n ($1 \leq s \leq 10, 0 \leq n \leq 99999999$), where n is the number to be displayed and s is the size in which it shall be displayed.

The input file will be terminated by a line containing two zeros. This line should not be processed.

4.2 Output

Output the numbers given in the input file in an LC-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, also for the last digit.) There has to be exactly one column of blanks between two digits.

Output a blank line after each number. (You will find a sample of each digit in the sample output.)

4.3 Sample Input

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

4.4 Sample Output

```
  --  --          --
 |  |  |  |  |  |
 |  |  |  |  |  |
  --  --  --  --
 | |  |  |  |  |
 | |  |  |  |  |
  --  --          --

 ---  ---  ---  ---  ---
 |  |  |  |  |  |  |
 |  |  |  |  |  |  |
 |  |  |  |  |  |  |
 ---  ---  ---
 |  |  |  |  |  |  |
 |  |  |  |  |  |  |
 |  |  |  |  |  |  |
 ---  ---  ---  ---
```

5 Interpreter (10033)

A certain computer has 10 registers and 1000 words of RAM. Each register or RAM location holds a 3-digit integer between 0 and 999. Instructions are encoded as 3-digit integers and stored in RAM. The encodings are as follows:

- 100 means halt
- 2dn means set register d to n (between 0 and 9)
- 3dn means add n to register d
- 4dn means multiply register d by n
- 5ds means set register d to the value of register s
- 6ds means add the value of register s to register d
- 7ds means multiply register d by the value of register s
- 8da means set register d to the value in RAM whose address is in register a
- 9sa means set the value in RAM whose address is in register a to the value of register s
- 0ds means goto the location in register d unless register s contains 0

All registers initially contain 000. The initial content of the RAM is read from standard input. The first instruction to be executed is at RAM address 0. All results are reduced modulo 1000.

5.1 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 to your program consists of up to 1000 3-digit unsigned integers, representing the contents of consecutive RAM locations starting at 0. Unspecified RAM locations are initialized to 000.

5.2 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.

The output from your program is a single integer: the number of instructions executed up to and including the halt instruction. You may assume that the program does halt.

5.3 Sample Input

```
1
299
492
495
399
492
495
399
283
279
689
078
100
```

000
000
000

5.4 Sample Output

16

6 Check the Check (10196)

6.1 The Problem

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....
..*.*...	...*....	..*.*...	..***..	..***...	..*..*..
.....	...*....	*...*..	*..*..**..*..
.....	...*....	*.....*	*..*..*

Remember that the knight is the only piece that can jumper 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).

6.2 The Input

There will be an arbitrary number of board configurations 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).

6.3 The 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).

6.4 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
```

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

6.5 Sample Output

Game #1: black king is in check.

Game #2: no king is in check.

Game #3: white king is in check.

7 Australian Voting (10142)

Australian ballots require that the voter rank the candidates in order of choice. Initially only the first choices are counted and if one candidate receives more than 50% of the vote, that candidate is elected. If no candidate receives more than 50%, all candidates tied for the lowest number of votes are eliminated. Ballots ranking these candidates first are recounted in favour of their highest ranked candidate who has not been eliminated. This process continues [that is, the lowest candidate is eliminated and each ballot is counted in favour of its ranked non-eliminated candidate] until one candidate receives more than 50% of the vote or until all candidates are tied.

7.1 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 first line of input is an integer $n \leq 20$ indicating the number of candidates. The next n lines consist of the names of the candidates in order. Names may be up to 80 characters in length and may contain any printable characters. Up to 1000 lines follow; each contains the contents of a ballot. That is, each contains the numbers from 1 to n in some order. The first number indicates the candidate of first choice; the second number indicates candidate of second choice, and so on.

7.2 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.

The Output consists of either a single line containing the name of the winner or several lines containing the names of the candidates who tied.

7.3 Sample Input

```
1
3
John Doe
Jane Smith
Sirhan Sirhan
1 2 3
2 1 3
2 3 1
1 2 3
3 1 2
```

7.4 Output for Sample Input

```
John Doe
```

8 Hartals (10050)

A social research organization has determined a simple set of parameters to simulate the behavior of the political parties of our country. One of the parameters is a positive integer h (called the *hartal parameter*) that denotes the average number of days between two successive *hartals* (strikes) called by the corresponding party. Though the parameter is far too simple to be flawless, it can still be used to forecast the damages caused by *hartals*. The following example will give you a clear idea:

Consider three political parties. Assume $h_1 = 3$, $h_2 = 4$ and $h_3 = 8$ where h_i is the *hartal parameter* for party i ($i = 1, 2, 3$). Now, we will simulate the behavior of these three parties for $N = 14$ days. One must always start the simulation on a Sunday and assume that there will be no *hartals* on weekly holidays (on Fridays and Saturdays).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Days														
	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
Party 1			x			x			x			x		
Party 2				x				x				x		
Party 3								x						
Hartals			1	2				3	4			5		

The simulation above shows that there will be exactly 5 *hartals* (on days 3, 4, 8, 9 and 12) in 14 days. There will be no *hartal* on day 6 since it is a Friday. Hence we lose 5 working days in 2 weeks.

In this problem, given the hartal parameters for several political parties and the value of N , your job is to determine the number of working days we lose in those N days.

8.1 Input

The first line of the input consists of a single integer T giving the number of test cases to follow.

The first line of each test case contains an integer N ($7 \leq N \leq 3650$) giving the number of days over which the simulation must be run. The next line contains another integer P ($1 \leq P \leq 100$) representing the number of political parties in this case. The i th of the next P lines contains a positive integer h_i (which will never be a multiple of 7) giving the *hartal parameter* for party i ($1 \leq i \leq P$).

8.2 Output

For each test case in the input output the number of working days we lose. Each output must be on a separate line.

8.3 Sample Input

```
2
14
3
3
4
8
100
4
12
15
25
40
```

8.4 Sample Output

```
5
15
```

9 Contest Scoreboard (10258)

Think the contest score boards are wrong? Here's your chance to come up with the right rankings.

Contestants are ranked first by the number of problems solved (the more the better), then by decreasing amounts of penalty time. If two or more contestants are tied in both problems solved and penalty time, they are displayed in order of increasing team numbers.

A problem is considered solved by a contestant if any of the submissions for that problem was judged correct. Penalty time is computed as the number of minutes it took for the first correct submission for a problem to be received plus 20 minutes for each incorrect submission received prior to the correct solution. Unsolved problems incur no time penalties.

9.1 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.

Input consists of a snapshot of the judging queue, containing entries from some or all of contestants 1 through 100 solving problems 1 through 9. Each line of input will consist of three numbers and a letter in the format

```
contestant problem time L
```

where L can be C, I, R, U or E. These stand for Correct, Incorrect, clarification Request, Unjudged and Erroneous submission. The last three cases do not affect scoring.

Lines of input are in the order in which submissions were received.

9.2 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.

Output will consist of a scoreboard sorted as previously described. Each line of output will contain a contestant number, the number of problems solved by the contestant and the time penalty accumulated by the contestant. Since not all of contestants 1-100 are actually participating, display only the contestants that have made a submission.

9.3 Sample Input

```
1
1 2 10 I
3 1 11 C
1 2 19 R
1 2 21 C
1 1 25 C
```

9.4 Sample Output

```
1 2 66
3 1 11
```