21st Annual Denison Spring Programming Contest
Granville, Ohio
27 February, 2010

Rules:

1. There are **six** questions to be completed in **four hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases unless otherwise noted.
7. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
8. All communication with the judges will be handled by the PC^2 environment.
9. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Bowling for Dullards

This bowling game is a little different from the usual ten pin bowling you might be use to. This game is played between two players who alternate turns. Here, you start with a row of n tightly packed pins. You both are very accurate bowlers and when you strike a pin you knock it over plus the pins immediately next to it. (This is required by the rules.) Note that there may be zero, one, or two pins immediately next to the pin you strike,, depending on what has happened earlier in the game. You alternate rolling the balls and the winner is the one knocking over the last pin. You must knock over some pin (and its neighbors) on each roll. The problem here is given an intermediate pin configuration, can you win given that your opponent makes the best possible rolls for him?

For example, consider the following configuration of pins at some point in the game. (`X`=pin, `O`=open spot)

```
XXOOOOOXOOOXXX
```

If we number the pins starting with 1 from the left, this configuration was realized by knocking over pin 4 (with neighbors 3 and 5), then pin 10 (with neighbors 9 and 11) and finally pin 6 (with neighbor 7). It's your roll. You can easily see you have a winning strategy if you aim at pin 13. This would leave

```
XXOOOOOXOOOOOO
```

Your opponent either knocks over the first two pins, or the pin standing by itself and then you can clean up. Note your opponent could not knock over just pin 1 or pin 2 as the rules require that adjacent pins must also fall.

Input

Input for each test case is on two lines. The first line contains a positive integer n ($n \leq 20$). The second line gives a string of n `X`'s and `O`'s (no spaces between the characters) indicating a position of a game in progress. It is your roll next. You are to determine if you have a winning strategy. There will be at least one `X` in the string. The test cases will be followed by a line containing 0.

Output

Your output should be either `Yes` or `No`, formatted as in the sample output.

Sample Input

```
14
XXOOOOOXOOOXXX
10
XOOXOOXXXX
0
```

Sample Output

```
Case 1: Yes
Case 2: No
```

Problem B: Why Jai Alai?

In a Jai Alai tournament 10 players are in a queue (players are numbered 1 through 10, front to back in the queue). The first two players play each other, with the loser going to the end of the queue. The winner is now at the front. This player now plays the next in line, with the loser going to the end and the winner remaining at the front to play the next player. This continues for many rounds. The question here is if you know who wins each match, either the current winner or the challenger, to find the order of the queue at the end of the tournament. (To start things off, we'll consider player 1 to be the current winner and player 2 to be the challenger.)

We'll use `C` to indicate the current winner again wins the round and `G` to indicate the challenger wins the current round. For example, after a round of 20 with the winners CCGCGGGCGCGGGCCCGGCG, the queue would be 10 3 1 4 6 7 5 9 2 8.

Input

Input for each test case will be on one line. A positive integer n, no larger than 100 will be first on the line, followed by a string of length n consisting of `C`'s and `G`'s. A line containing 0 will follow the last test case.

Output

For each test case print the 10 player numbers in the order they appear in the queue, front to back, formatted as shown in the sample output.

Sample Input

```
6 CGCCGC
20 CCGCGGGCGCGGGCCCGGCG
10 CCGGCCGGCC
0
```

Sample Output

```
Case 1: 6 8 9 10 2 1 4 5 3 7
Case 2: 10 3 1 4 6 7 5 9 2 8
Case 3: 9 3 1 4 6 7 5 8 10 2
```

Problem C: Doctor Evil's Escape

Austin Powers is closing in on Doctor Evil who is holed up in his mountain-top lair on an island in the arctic. (At this point, insert your pinkie in the corner of your mouth. Give an maniacal laugh if you wish.) But Doctor Evil has an escape plan using his sled, which is also a submarine. The plan is to slide down the snow-covered mountains into the ocean. Unfortunately he has not entirely tested things and does not know where he will emerge or even if he will make it to the ocean.

The steering on his sled/submarine is a little limited and can only go in the direction of the four compass points (east, west, north, south). Further, he can only turn every 100 feet. You (but not Doctor Evil) have a map with the elevations every 100 feet. (How convenient!) Doctor Evil has decided that the fastest way down is probably the best and so every 100 feet, he either goes straight ahead, turns to his immediate left, or turns to his immediate right, according to which direction offers the most drop in elevation. If there is a tie, he will always go straight ahead, if that is one of the steepest directions, or otherwise will turn left, if both left and right turns are steepest. His sled will only move if there is a negative change in elevation. Of course, he may find himself at the bottom of a valley — too bad for him.

The island he's on is square and your map of the island has integer elevations. Your job is to figure out where Doctor evil ends up. He will either exit the island to the north, east, south or west, or get trapped in a valley.

Input

Input for each test case will consist of multiple lines. The first line will be a positive integer n ($n \leq 1000$) indicating the number of elevation points for a side of the island. There follows n lines, each containing n positive integers, each no more than 100000, giving the grid elevations of the island. The northern most line of elevations, is given first, west to east. The next line gives the elevations to the immediate south, and so on. (The ocean has elevation 0.) A line with 0 will follow the test cases. Doctor Evil's lair is on the highest point on the map (which will be unique and not on the edge of the island). His sled initially will take the steepest direction, which will also be unique.

Output

Output for each test case should be either `north`, `east`, `south`, `west`, or `trapped in valley`, accordingly, formatted as in the sample output.

(over)

Sample Input

```
20
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
10 10 10 10 10 35 35 18 35 35 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 35 18 20 22 25 22 10 10 10 10 10 10 10 10 10
10 10 10 10 10 35 15 18 35 30 35 10 10 10 10 10 10 10 10 10
10 10 10 10 10 35 14 35 35 40 35 10 10 10 10 10 10 10 10 10
10 10 10 10 10 35 13 35 35 35 35 10 10 10 10 10 10 10 10 10
35 35 35 35 35 35 12 35 10 10 10 10 10 10 10 10 10 10 10 10
5 6 7 8 9 10 11 35 35 35 10 10 10 10 10 10 10 10 10 10
35 35 35 35 35 35 35 35 10 10 10 10 10 10 10 10 10 10 10 10
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25 25
10
1 1 1 2 2 1 1 1 2 2
1 2 2 2 2 2 3 4 5 2
1 4 5 6 7 6 5 4 3 2
1 2 3 4 5 4 3 3 4 2
1 2 3 5 6 8 7 6 5 4
1 2 3 5 6 7 7 7 5 4
2 2 3 4 5 6 6 5 4 3
2 3 4 5 4 3 4 5 6 4
2 3 4 5 4 3 2 2 2 1
1 2 3 2 3 2 3 2 2 1
0
```

Sample Output

```
Case 1: west
Case 2: trapped in valley
```

Problem D: Please, Take A Seat

The TransOhio Airways (TOA) exclusively flies Long and Skinny Jets (LSJ's). LSJ's have one seat per row, numbered 1 through n, for various values of n. The aisle is on one side of the plane and the seats are on the other making plane list to one side, which in turn makes it hard to fly — but that's another problem. TOA flights are cheap and so very popular. Indeed, every flight sells out. Passengers are given their seat numbers and then enter the plane from the front in some haphazard order and proceed to their seats. Each passenger goes to her row immediately, unless blocked by another passenger. Once a passenger gets to her row, she takes one time period (about 20 seconds) to get seated. The passengers behind her who were blocked can then move immediately to their seats, etc. When blocked, a passenger occupies one row and so might in turn prevent others behind from getting to their rows. Your job is to determine how long it takes to get everyone seated.

For example, let's suppose in a 5-seat plane, the passengers enter the airplane in order 4, 2, 5, 1, 3. Passenger 4 immediately moves to her seat, as does passenger 2 behind her. But passenger 2 is blocking 5, who is standing in row 1 and so is blocking 1. In the next time period, now that 4 and 2 are seated, 5 is free to move to her row, as is 1, who is blocking 3. Passengers 5 and 1 are seated and in the next time period 3 moves to her row and is seated. Thus it takes a total of 3 time periods to get everyone seated.

Input

Input for each test case is on two lines, the first of which contains the positive integer n ($n \leq 100$) which is the number of seats on the plane. The second line is a permutation of the first n positive integers, giving the order the passengers enter the plane. A line with 0 follows the last test case.

Output

For each test case output the number of time periods it takes to seat all the passengers, formatted as shown in the sample output.

Sample Input

```
5
4 2 5 1 3
10
7 8 5 6 9 4 3 10 1 2
0
```

Sample Output

```
Case 1: 3
Case 2: 4
```

Problem E: The Simple Life

Nearly everyone has looked at the Game Of Life, an example of cellular automata. Here we'll look at a simpler version; one you might call Simple Eternal Life, but Not Too Crowded. This world is played out on a recangular grid of cells, each cell is either alive or dead. All cells but one are initially dead. Each cell has exactly 4 neighbors (N, E, S, and W). At each generation, a dead cell comes alive in the next generation if exactly one of its neighbors is now alive. All births occur simultaneously at each generation and once a cell becomes alive, it stays alive.

At the genesis, there is but one cell. The world looks like the following (0 =dead, 1 = alive) after the 1st, 2nd and 3rd generation. (All other cells are dead.):

```
                                0 0 0 1 0 0 0
          0 0 1 0 0             0 0 1 1 1 0 0
0 1 0     0 0 1 0 0             0 1 0 1 0 1 0
1 1 1     1 1 1 1 1             1 1 1 1 1 1 1
0 1 0     0 0 1 0 0             0 1 0 1 0 1 0
          0 0 1 0 0             0 0 1 1 1 0 0
                                0 0 0 1 0 0 0
```

You are to compute the number of alive cells after each generation.

Input

Input for each test case is a single integer n ($n \leq 100$) on a line. A line containing zero follows the last test case.

Output

For each test case, output the number of alive cells after the requested generation, using the format in the sample output.

Sample Input

```
1
2
3
0
```

Sample Output

```
Case 1: 5
Case 2: 9
Case 3: 21
```

Problem F: Booklet Making

You work for a printing company that makes booklets. A customer sends you the pages of their booklet, along with the table of contents (TOC). You arrange this matter on sheets where four pages are printed onto each sheet, 2 pages on the front and 2 on the back, so that when the sheets are stacked and folded, the pages are in order.

The booklet will always start with the front matter which consists of the Title Page, followed by a blank page, followed by the TOC, then the chapters of the book. Chapters always start on the right-hand page with Chapter 1 starting on page 1. So, a blank page may need to be inserted just before the start of some chapters. The numbers of the blank pages are not printed on the page. The front matter pages are numbered -1, -2, etc, with the Title Page having number -1.

For example, suppose a customer has a one page TOC and 3 chapters of length 5, 6, and 6 pages (in that order). The front matter would take up 4 pages (Title Page, blank page, TOC, blank page), Chapter 1 takes 5 pages, plus a blank (so that Chapter 2 starts on an odd-numbered page). Chapter 2 starts on page 7 and takes takes 6 pages. Chapter 6 starts on page 13 and takes 6 pages, for a total of 22 pages. The chapters of the book run from page 1 through page 18. You'll need 6 sheets to print this book (4 pages to a sheet) so the last two pages will be blank.

Now to print this booklet so that it can be folded and the pages will be in order, the outside sheet will need to have the last page (blank) on the left and page -1 (the Title Page) on the right on the "top" side and a blank on the left (the page that follows the Title Page) and a blank on the left (next-to-last blank page) on the right on the "bottom" side. (We look at the bottom side as we flip over the sheet.) The top of the next sheet has page 18 on the left and page -3 on the right (the TOC), while the bottom has page -4 on the left and page 17 on the right. And so on.

Given information about a customer's order, you are to find the layout of each sheet in the customer's book.

Input

Input for each test case will be a positive integer n, indicating the number of chapters in the booklet, followed by $n + 1$ positive integers giving the number of pages in the TOC, followed by the number of pages in each of the chapters. Your booklet will have no more than 40 pages total. A line containing 0 will follow the last test case.

Output

You are to output the layout of all the sheets in the booklet from outermost sheet to innermost, on one line. You should give the pages on each sheet in the format `L/R,l/r` where `L` and `R` are the left and right pages of the top side and `l` and `r` are the left and right pages of the bottom side. If the page is blank, do no print the page number, but print `B`. There are to be no spaces used in this. But separate info about the sheets with a space. See the format in the sample output.

Sample Input

```
3 1 5 6 6
4 2 4 4 4 3
0
```

Sample Output

```
Case 1: B/-1,B/B 18/-3,B/17 16/1,2/15 14/3,4/13 12/5,B/11 10/7,8/9
Case 2: B/-1,B/15 14/-3,-4/13 12/1,2/11 10/3,4/9 8/5,6/7
```