Question 1 — Data Entry

Your company has been fortunate to have one employee devoted only to data entry: Doris. She loves her job and is the fastest around. Fortunately for the company, business is booming. Unfortunately, the company hired another person for data entry: Boris. He's almost as fast as Doris. Unfortunately, they are stubborn and completely disagree about the proper way to enter data.

Doris insists on knowing exactly how many pieces of data she needs to type in. "How can you enter data at peak speed if you don't pace yourself? And how can you pace yourself if you don't know how much data there is?"

Boris insists on jumping right into the data and then entering a sentinel value when he's finished. "Start faster. Finish faster."

Your job is to write a data entry program that will accommodate both Doris and Boris.

Program input: The first line indicates the number of data sets. Each data set will consist of a set of names, with each name on a separate line. Each data set will contain at least one name. Also, if it is one of Doris's data sets, then before the set of names there will be a number indicating how many names are in the data set. If it one of Boris's data sets, then at the end of the set of names there will be a line containing the number 0 (the sentinel value). Example input:

```
4
3
Alice
Bob
Charlie
Dierdre
Edie
Frank
Georgina
Harry
0
Ingrid
Jerome
Kyle
Lyle
Myles
Nial
0
2
Opal
Phyllis
```

Program output: The program should output the number of names entered by Doris followed by the number of names entered by Boris. Label the numbers and put them on separate lines. Example output (corresponding to the example input shown above):

```
Doris: 5
Boris: 11
```

Question 2 — Odometer Factory

Your company is the premier manufacturer of custom digital car odometers. Your customers include car companies from all over the world. For some reason, many of your customers need odometers that count in bases other than base 10. For example, Fredonia (a tiny country sandwiched between its larger neighbors Liechtenstein and Lesotho) uses base 5 for its car odometers, while New Kerala (an island nation in the Weddell Sea of Antarctica) uses base 9. Creating a new version of your odometer software for each different base is too time consuming, so you decide to create universal odometer software that can handle any of the different bases your customers require.

Program input: The program input consists of one or more lines. Each line contains four integers (consisting of the digits 0 through 9): the base, the number of digits, the initial reading, and the number of times to increment. The base is between 2 and 10 inclusive. The number of digits indicates how many digits the odometer can display, and will be between 4 and 20 inclusive. The initial reading is the initial odometer value, *in the specified base*. The number of times to increment specifies how many times the odometer should be incremented. With the exception of the initial odometer reading, all of the integer values on a line are base 10 values. As a special case, if a line begins with a base of -1, that indicates the end of input, and the program should exit.

Example input (showing three lines and the end of input marker):

```
5 6 013143 7
9 4 1747 4
3 5 22221 2
-1
```

Program output: For each input line (other than the end-of-input marker), the program should write a single line of output showing an incremented odometer reading, in the specified base. In other words, the output lines show what happens when the odometer's initial value is incremented by 1 the specified number of times. Example output (showing the expected output for the input shown above):

```
013144
013200
013201
013202
013203
013204
013210
1748
1750
1751
```

```
1752
22222
00000
```

Hints and specifications:

- Each odometer reading must have exactly the specified number of digits.
- When an odometer displaying the maximum value is incremented, it "rolls over" so that every digit is 0 (zero)

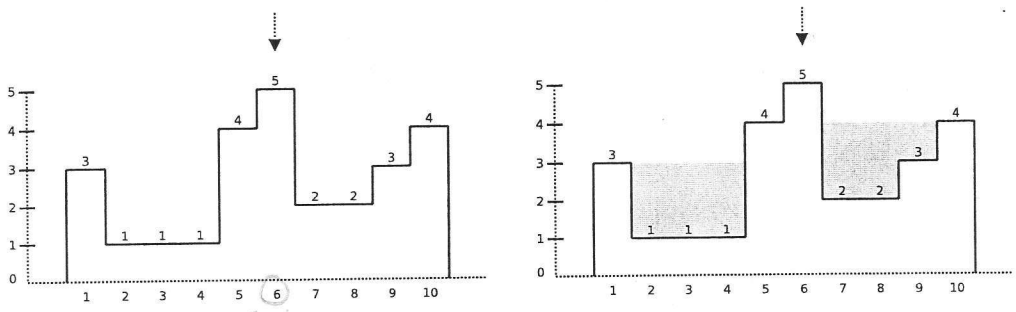
Question 3 — Falling Water

You are a designer of two-dimensional water sculptures. Your sculptures consist of a series of rectangular towers of varying heights. Each tower height is a positive integer. Each tower is the same width. Adjacent towers are pressed tightly together to form a water-tight seal. As a special case, a tower of height 0 (zero) is not a tower at all, but instead is a gap that exposes an infinite void. Also, there are voids to the left of the leftmost tower and to the right of the rightmost tower.

Each sculpture has water poured from above onto one of the towers. The water flows to the left and the right from the roof of the tower onto which it is poured. If a tower roof has water flowing on it, the water will flow onto the roofs of neighboring towers if they are at the same height or lower. If a sequence of one or more tower roofs with flowing water have neighbors at both ends that are above the level of the water, the water will rise to form a pool. The level of water in a pool is the height of the lower of the two towers that form the walls at either end of the pool. Note that water from a pool can flow onto the roof of a tower on one end of the pool, if the tower's roof is at the same height as the water level in the pool. If a tower whose roof has flowing water is adjacent to a void, the water falls into the void on that side of the tower. Pools never form above voids.

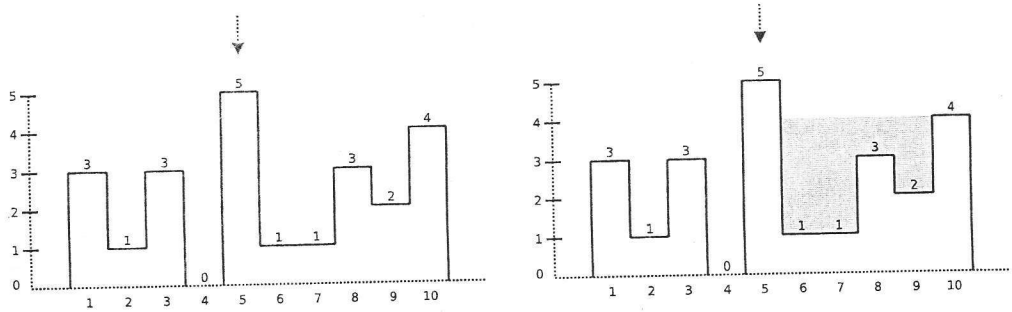
In order to get an idea of what the sculpture will look like, you decide to write a program to determine, based a configuration of towers and where water will be poured, what the sculpture will look like when pools form.

Here is an example sculpture (left is the "plain" sculpture, right is the sculpture after pools have formed):



The towers are numbered 1–10 from left to right. The numbers above each tower indicate roof height. The arrow above tower 6 indicates that water is being poured onto the top of that tower. This particular sculpture has no voids (other than the implicit voids to the left of tower 1 and the right of tower 10.)

Here is a second example sculpture (left "plain", right with pools):



Note that because position 4 is a void, no pools form to the left of tower 5.

Program input: The program takes a series of lines. Each line is a sequence of integers describing one sculpture. The first integer indicates how many horizontal positions there are. Next, a series of integers indicates the height of the towers at each position (with 0 indicating a void.) The final integer indicates the position of the tower on which water is poured.

As a special case, if the first integer on a line is `-1`, that means the end of input for the program.

Example input (the two example sculptures, and the end-of-input marker):

```
10 3 1 1 1 4 5 2 2 3 4 6
10 3 1 3 0 5 1 1 3 2 4 5
-1
```

Program output: For each input line (except the end-of-input marker), the program should print one output line. An output line is a series of integers showing the water level at each position.

Example output (corresponding to the example input):

```
3 3 3 3 4 5 4 4 4 4
0 0 0 0 0 5 4 4 4 4
```

Hints and specifications:

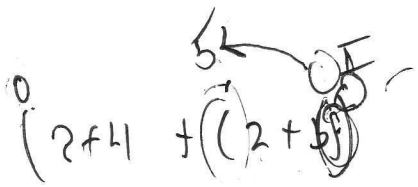
- Hint: create some test cases of your own.
- There will not be more than 20 positions in any sculpture. The maximum tower height is 20.
- You should not assume that each sculpture described in the input file will have the same number of positions.
- The water level for a tower that has water flowing on its roof, but which is not submerged in a pool, should be equal to the height of the tower.
- The water level of a tower that does not have water flowing on its roof and is not submerged should be 0 (zero).

Question 4 — Expression Evaluator

Write a program that evaluates infix expressions involving integer values. The evaluator should support the following operators: + (addition), - (subtraction), and * (multiplication), and / (division). It should also allow parentheses to be used for grouping. The evaluator should use standard 32-bit signed integers (i.e., the standard `int` data type in Java and C/C++.) Division is integer division (meaning that division operations should discard any remainder.)

Program input: The input will consist of one or more lines, such that each line specifies one infix expression. There is guaranteed to be exactly one space character between each token (integer value, operator, or parenthesis.) As a special case, a line containing the text `quit` indicates that there is no more input, and the program should exit. You can assume that every expression will be valid. Example input:

```
4
( 4 )
23 / 3
3 + 4 * 5
( 3 + 4 ) * 5
25 - 9 - 3
quit
```



Program output: For each input expression, the program should print a single line containing the integer result of evaluating the expression. Example output (corresponding to the example input shown above):

```
4
4
7
23
35
13
```

Hints and specifications:

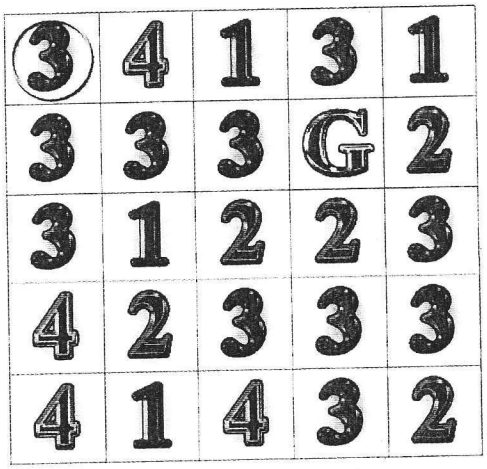
- The + and - operators have equal precedence
- The * and / operators have equal precedence, and have higher precedence than + and -
- All of the operators are left associative: e.g., `25 - 9 - 3` means $(25 - 9) - 3$

This page is intentionally blank

Question 5 — Off the Rook

Create a program that, for each of a sequence of specified Rook Jumping Mazes, prints the number of moves in a shortest solution, or "`No solution.`" if no solution exists.

Here is an example of a Rook Jumping Maze:



A Rook Jumping Maze is defined as an m-by-n grid of jump numbers, a start cell (circled above), and a goal cell (marked "G" above). According to each cell's jump number, one may move that exact number of cells horizontally or vertically in a straight line. One may not wrap around edges, change directions mid-move, or move diagonally. One goal of a Rook Jumping Maze is to find the shortest path, i.e. the minimum number of moves, from the start cell to the goal cell. For the example above, there is a minimum of 13 moves: DRLUDLRULLRDU, where R, L, U, D stand for Right, Left, Up, and Down, respectively.

Program input: The input consists of a series of maze specifications followed by single line with a terminating zero (0).

Each maze specification begins with a line of space-separated integers providing:

- Number of rows (m)
- Number of columns (n)
- Start row
- Start column
- Goal row
- Goal column

Start and goal cells coordinates are zero-based, i.e. the first row and first column are at (0, 0).

What follows is a grid of jump numbers, i.e., m lines each representing a row of jump numbers having n space-separated integers. The integer in the ith row and jth column is the jump number for cell (i, j).

Example input:

```
5 5 0 0 1 3
3 4 1 3 1
3 3 3 0 2
3 1 2 2 3
4 2 3 3 3
4 1 4 3 2
5 5 0 0 4 4
3 3 2 4 3
2 2 2 1 1
4 3 1 3 4
2 3 1 1 3
1 1 3 2 0
1 30 0 0 0 10
15 11 19 21 7 1 23 12 17 1 0 10 4 9 12 1 10 2 6 10 3 7 4 6 1 3 24 25 23 2
0
```

Program output: For each solvable maze, print a single line with an integer indicating the number of moves in the shortest solution. For each unsolvable maze, print a single line with the string "`No solution.`"

Example output (corresponding to example input shown above):

```
13
No solution.
20
```

Gregorian Calendar

Question 6 — Prime Birthdays

Prime numbers are cool. And birthdays are cool. Even cooler still are birthdays where your age becomes a prime number. And for ultimate coolness, nothing beats going to a birthday party where everyone's age is prime.

Your task is to write a program that will take a set of birthdays (of your family and friends, natch), the start date, and a number of years, and compute the earliest day equal to or later than the start date when (a) it is someone's birthday, and (b) everyone whose birthdays were specified has a prime age, as long as the date meeting criteria (a) and (b) occurs on or before the date computed by adding the specified number of years to the start date. The resulting date, if one is found, is the next opportunity for you and your peeps to have an ultimately cool prime number birthday party where everyone at the party has a prime age. (Note that it doesn't matter *whose* birthday it is, only that everyone at the party has a prime age.) Sadly, depending on the set of birthdays you use, you might not find a viable date for a prime birthday party, but c'est la vie.

Program input: Each line of input consists of

- The number of birthdays
- The birthdays in DD-MM-YYYY format
- The start date in DD-MM-YYYY format
- The number of years

All items are separated with one or more space characters. As a special case, if a line starts with the number 0, then there is no more input, and the program should exit. Example input:

```
3 20-06-1994 09-09-1981 02-01-1992 01-09-2014 5
3 20-06-1994 09-09-1981 02-01-1992 01-09-2014 10
4 01-12-1991 29-02-1996 10-07-1973 12-01-1980 24-10-2015 40
5 28-09-1986 28-04-1971 02-10-1980 13-06-1990 22-10-1976 01-01-1991 1000
5 28-09-1986 28-04-1971 02-10-1980 13-06-1990 22-10-1976 29-04-1994 1000
6 28-09-1986 28-04-1971 02-10-1980 13-06-1990 22-10-1976 25-05-1984 01-01-1991 1000
0
```

Program output: For each input line, the program should print either

- a date (in DD-MM-YYYY format), followed by the ages of each person whose birthday was specified, or
- "`No date found.`"

The date printed should be the earliest day on or after the specified start day when it is someone's birthday and everyone whose birthday was specified has a prime age, as long as it is no more than the specified number of years from the start date. If there is no such

date, the output "`No date found.`" should be printed. Example output (corresponding to the example input shown above):

```
No date found.
20-06-2023 29 41 31
01-03-2033 41 37 59 53
28-04-1994 7 23 13 3 17
No date found.
No date found.
```

Hints and specifications:

- Each line will have no more than 10 birthdays
- January, March, May, July, August, October, and December have 31 days
- April, June, September, and November have 30 days
- February has 29 days in leap years, 28 days otherwise
- Years that are either (a) divisible by 4 and not divisible by 100, or (b) divisible by 400 are leap years
- To add n years to a date, just add n to the date's year; the only special case is when the original date is February 29th and adding n to the original date's year results in a year that is not a leap year, in which case the resulting date should be changed from February 29th to March 1st
- If a birthday occurs on February 29th, then in non-leap years (where there is no February 29th) the birthday should be treated as occurring on March 1st
- A prime number is an integer 2 or greater divisible only by 1 and itself