Sixth Annual Quinnipiac University Programming Competition for High School Students

Quinnipiac University

April 24, 2010

Instructions

1. Submit a program by attaching all necessary files to an e-mail to `cscjudge`. Include the number of the problem you are solving in the subject line of your e-mail.

   Questions may also be addressed to `cscjudge`; put the word "Question" in the subject line, along with the number of the problem you are asking about if your question pertains to a particular problem.

2. You may assume the input is formatted as shown in the problem, and your program must work on such input. You may assume there are no tabs in the input.

3. Your feedback on a submitted problem will be e-mailed in a reply and will be limited to one of the following:

   (a) COMPILE ERROR: The program did not compile

   (b) PROGRAM NOT RUNNABLE: The program compiled, but did not run as a stand-alone program.

   (c) CRASH: The program crashed or threw an exception when it was run

   (d) TIMED OUT: The program did not finish in a reasonable amount of time

   (e) INCORRECT OUTPUT: Either the answer is incorrect, or it is not presented in the form specified in the problem.

   (f) ACCEPTED

   If your program crashes, produces incorrect output, or times out, no information will be given as to the input that caused the problem.

1 Inversions in a sequence

Given a sequence of characters, $(c_1, c_2, \ldots c_k)$, we say the pair (c_i, c_j) is an *inversion* if $i < j$ but c_i comes after c_j in the alphabet. For example, in the sequence $(C,\ E,\ B,\ F,\ A,\ D)$, we can use the following table to list the inversions:

First Character	Inversions
($\boxed{C}$, E, **B**, F, **A**, D)	$(C,\ B)$, $(C,\ A)$
(C, $\boxed{E}$, **B**, F, **A**, **D**)	$(E,\ B)$, $(E,\ A)$, $(E,\ D)$
(C, E, $\boxed{B}$, F, **A**, D)	$(B,\ A)$
(C, E, B, $\boxed{F}$, **A**, **D**)	$(E,\ A)$, $(E,\ D)$
(C, E, B, F, $\boxed{\text{A}}$, D)	none
(C, E, B, F, A, $\boxed{\text{D}}$)	none

Write a program which counts the number of inversions in a sequence of uppercase letters. The input begins with a single integer n on a line by itself representing the number of characters, followed by a line with n uppercase letters, separated by spaces. The program then prints the number of inversions in the sequence. The input corresponding to the example above would be

```
6
C E B F A D
```

and the program should print the number 8 given this input. You may assume that $2 \leq n \leq 26$, but you should not assume that the letters are distinct. If the same letter appears twice in the input, that does not count as an inversion. As another example, if the input is

```
10
I N V E R S I O N S
```

there is one inversion with the first I as the first letter, two beginning with the first N, seven beginning with the V, three beginning with the R, three beginning with the first S, and one beginning with the O. The E, the second I, the second N, and the second S are not the first letter for for any inversions. Therefore, the corresponding output should be 17.

2 Walking about town

In the city of Quinnip, avenues run East/West and are numbered consecutively beginning with First Avenue along the southern edge of the city, and streets run North/South and are numbered beginning with First Street on the western edge of the city.

Bob Cat, who lives at the corner of First Avenue and First Street, likes to walk through the city, but he does not like to follow the same route twice. Each day, he tosses a coin several times; each time it comes up heads, he walks north, and when it lands tails, he walks east. At first he walks one block in the chosen direction, but he often gets bored of this routine (sometimes even before he starts out). When this happens, he rolls a six-sided die, and where the die lands determines the number of blocks he walks, north or east, for each subsequent coin toss. He may roll the die several times during the course of his walk.

For example, if he tosses heads, then tails, then tails again, he walks one block north to second avenue, then two blocks east to third street. If he then rolls his die and it lands on a `4`, he will now walk four blocks in the prescribed direction every time he tosses the coin. Thus, if his next two tosses are heads, he walks north to tenth avenue and third street, at which point he may decide to roll the die again. He never walks more than a total of fifty blocks, even if his last toss required him to continue, though he may stop before fifty blocks.

Write a program which takes as input a string representing a sequence of coin tosses and die rolls, and gives the intersection he ends on after the given sequence. Within this string, the characters '`H`' and '`T`' represent coin tosses of heads and tails, respectively, and the digits '`1`' through '`6`' represent rolls of the die. There are no other characters in the input (not even spaces), and all characters are on the same line. Your program should print out the avenue number followed by the street number where he ends up.

The following table gives examples of input along with the corresponding output.

Input	**Output**	**Explanation**
`HTT4HH`	`10 3`	This is the example above
`HTHHT6HTTH35THH4TH`	`28 24`	Once he reaches the fiftieth block, he stops.
`16665362121`	`1 1`	He kept rolling the die, but he never tossed the coin, so he never moved anywhere.

You may assume that the total number of coin tosses and die rolls is at most fifty.

3 Composing Permutations

A *permutation* of the set $\{1, 2, 3, \ldots, n\}$ is a function that uses the numbers from 1 to n as input and as output, and in which every number appears once in the output. We usually use Greek letters such as σ (sigma) and τ (tau) to represent permutations. For example, one permutation of $\{1, 2, 3, 4, 5\}$ is the function

$$\sigma(1) = 3 \quad \sigma(2) = 5 \quad \sigma(3) = 1 \quad \sigma(4) = 4 \quad \sigma(5) = 2$$

Another way to represent this permutation is with *two-line notation*:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 4 & 2 \end{pmatrix},$$

or we can simply omit the first line and use *one-line notation*:

$$\sigma = (\ 3 \quad 5 \quad 1 \quad 4 \quad 2\ )$$

Given two permutations σ and τ of the set $\{1, 2, 3, \ldots, n\}$, we can create a new permutation, $\sigma \circ \tau$, called the *composition of σ and τ*, by defining $(\sigma \circ \tau)(i) = \sigma(\tau(i))$. For example, if σ and τ are given in one-line notation by

$$\sigma = (\ 3 \quad 5 \quad 1 \quad 4 \quad 2\ )$$
$$\tau = (\ 4 \quad 1 \quad 2 \quad 3 \quad 5\ )$$

then to compute $\sigma \circ \tau$, we have $(\sigma \circ \tau)(1) = \sigma(\tau(1)) = \sigma(4) = 4$; $(\sigma \circ \tau)(2) = \sigma(\tau(2)) = \sigma(1) = 3$; and so on. We find that

$$\sigma \circ \tau = (\ 4 \quad 3 \quad 5 \quad 1 \quad 2\ ).$$

Notice that order matters; we do not necessarily know that $\sigma \circ \tau = \tau \circ \sigma$. For example, with the above permutations σ and τ, we have

$$\tau \circ \sigma = (\ 2 \quad 5 \quad 4 \quad 3 \quad 1\ ) \neq \sigma \circ \tau.$$

Write a program which computes the result of composing a sequence of t permutations of $\{1, 2, \ldots, n\}$. The program takes the integers n and t, each on a line by itself, followed by t permutations of $\{1, 2, \ldots, n\}$ expressed in one-line notation as a line of n integers separated by spaces. If these permutations, in order, are σ_1, σ_2, and so on up to σ_t, then your program should compute the composition

$$\sigma_t \circ (\sigma_{t-1} \circ \cdots \circ (\sigma_2 \circ \sigma_1)).$$

The following table gives some examples:

Input	5 2 4 1 2 3 5 3 5 1 4 2	5 2 3 5 1 4 2 4 1 2 3 5	9 3 4 3 9 1 7 2 5 8 6 6 2 9 8 4 1 3 7 5 2 7 5 8 9 3 1 6 4
Output	4 3 5 1 2	2 5 4 3 1	6 4 9 3 5 7 8 1 2

In the last example, we have $\sigma_3 \circ (\sigma_2 \circ \sigma_1(1)) = \sigma_3(\sigma_2(\sigma_1(1))) = \sigma_3(\sigma_2(4)) = \sigma_3(8) = 6$, and similarly for $\sigma_3 \circ (\sigma_2 \circ \sigma_1(i))$ for every other i. You may assume $2 \leq n \leq 9$ and $1 \leq t \leq 9$.

4 Battleship

The game of Battleship is played on a rectangular grid (usually 10×10). The rows of the grid are indexed by letters starting from 'A' and the columns are indexed by numbers starting from 1.
One player places a collection of ships on the grid. Each ship occupies a fixed number of squares on the grid (one or more), and those squares must be consecutive in one direction, either horizontally in a single row, or vertically in a single column, and the ships may not overlap. The other player then takes "shots" at these ships, calling out the letter and number of the targeted square. The first player then responds by saying that the shot was either a "hit" if a ship was on the square or a "miss" if the square was empty. The game ends once all ships have been sunk, i. e. once the second player has fired at all squares where a ship has been placed. **At this point, further shots are ignored.**
The first player keeps track of where the ships are positioned and which squares were hit on each ship, and the second player keeps track of all shots fired, both hits and misses.
Write a program which gives the state of the game after the ships have been placed and zero or more shots have been fired.

The input

The first line gives the number of rows m and the number of columns n on the grid. The next line gives the number of ships, k, in the game. The numbers m, n, and k are all between 1 and 20, inclusive.
The next k lines each represent a ship. They begin with an upper- or lowercase letter that will be used to represent that ship, followed by three tokens that represent the placement of that ship: a letter followed by two numbers represent a ship placed horizontally; a number followed by two letters represent a ship placed horizontally.
The rest of the input represents a sequence of shots. The first of these lines gives the total number of shots, and every other line gives the letter of the row and the number of the column of a single shot with at least one space in between. You may assume all shots are distinct.
For example, the input might be:

```
10 12
3
c B 5 7
D 10 G H
z E 4 4
4
J 11
C 4
H 10
B 6
```

This specifies that we are playing on a 10×12 grid; there are three ships: the ship labeled `c` occupies the squares `B5`, `B6`, and `B7`, the ship `D` occupies `G10` and `H10`, and `z` occupies the single square `E4`; and four shots are fired: at `J11` (a miss), `C4`, (a miss), `H10` (hits `D`), and `B6`, (hits `c`).

The output

Your program should produce as output two $m \times n$ grids, separated by a single blank line. Each grid shows the information tracked by the corresponding player. Use the following characters on each grid:

First grid		Second grid	
Status of square	**Character**	**Status of square**	**Character**
Ship hit	`*`	**Hit**	`*`
Ship not hit	Label of ship	**Miss**	`@`
No ship	**.**	**No shot fired**	**.**

Notice that the first player does not keep track of missed shots, and the second player does not know what ships he has hit, and does not have any information about squares where he has not fired a shot.
For example, given the sample input of the previous page, the output should be as follows:

```
. . . . . .
. . . . C * C . . . . .
. . . . . .
. . . . . .
. . . Z . . . . . .
. . . . . .
. . . . . D . .
. . . . . * . .
. . . . . .
. . . . . .

. . . . . .
. . . . . * . . . . .
. . . @ . . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . * . .
. . . . . .
. . . . . @ .
```

Note: Ship labels are case sensitive; a ship labeled `D` is not the same as a ship labeled `d`.

5 Signed Ternary Numbers

Typically we represent a number in base 10, or decimal notation, which means we use ten digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, and we use a system of place values to represent arbitrarily large numbers. For example, when we write 4905, that is a shorthand notation for

$$4905 = 4 \cdot 10^3 + 9 \cdot 10^2 + 0 \cdot 10^1 + 5 \cdot 10^0.$$

Other number systems are possible; for example, computers typically use a base 2 (binary) system with the digits 0 and 1. This problem and the next one consider a variation of a base 3 (ternary) number system.

The typical ternary system uses the digits 0, 1, and 2, but we can represent numbers equally well using digits to represent -1, 0, and $+1$ together with powers of 3. We use the symbols `-`, `0`, and `+` as ternary digits. We call this a *signed ternary system*. Then, for example, the string `+-0--0+` represents the number

$$\texttt{+-0--0+} = 1 \cdot 3^6 + (-1) \cdot 3^5 + 0 \cdot 3^4 + (-1) \cdot 3^3 + (-1) \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 = 729 - 243 - 27 - 9 + 1 = 451.$$

In this system, if we count from 0 to 9, we can use the following table:

Decimal	0	1	2	3	4	5	6	7	8	9
Signed Ternary	0	+	+-	+0	++	+- -	+-0	+-+	+0-	+00

Negative numbers are the same as the corresponding positive number, except that every `+` digit is changed to a `-` and *vice versa*. For example, `-+0++0-` is the complement of our previous example, so its value is -451.

Write a program that converts a collection of numbers represented using this signed ternary system into their decimal equivalents. The input for the program begins with a (normal) decimal number n on a line by itself, representing the number of strings to convert, followed by a collection of n signed ternary strings, each on a line by itself. The program should output the decimal value of each of these signed ternary strings, each on a line by itself. For example, if the input is

```
5
0
+-+
-+-
+-0--0+
-+0++0-
```

the output should be

```
0
7
-7
451
-451
```

You may assume there are at most 100 input strings, each with at most sixteen characters.

6 Decimal to Signed Ternary

Write a program that converts a collection of decimal integers to their signed ternary representations. The input for the program begins with a number n on a line by itself, representing the number of integers to convert, followed by a collection of n decimal numbers, each on a line by itself. The program should output the signed ternary representation of each of these decimal numbers, each on a line by itself. For example, if the input is

```
5
0
7
-7
451
-451
```

the output should be

```
0
+-+
-+-
+-0--0+
-+0++0-
```

You may assume there are at most 100 numbers, and these all require at most sixteen characters to represent them.

7 The $3n + 1$ problem

Consider the function $f(n)$ defined by

$$f(n) = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases}$$

Given a positive integer x_i, we define the integer x_{i+1} by

$$x_{i+1} = f(x_i).$$

For example, if $x_0 = 3$, then $x_1 = 3 \cdot 3 + 1 = 10$, and $x_2 = \frac{10}{2} = 5$. We compute subsequent values of x_i according to the following table:

i	0	1	2	3	4	5	6	7
x_i	3	10	5	16	8	4	2	1

Given an arbitrary starting integer x_0, it is unknown whether every such sequence eventually reaches 1 (after which it would forever repeat the sequence 4, 2, 1).
Write a program which takes no input, and creates a 64×64 grid of characters in which the character in row i and column j depends on the first value of k for which $x_k = 1$ when $x_0 = 64 * (i - 1) + j$. The character is given by the following table; each digit other than 0 represents a block of sixteen numbers.

k	**Character**
0	0
1–16	1
17–32	2
33–48	3
49–64	4
65–80	5
81–96	6
97–112	7
113–128	8
129–144	9
145 or more	$\star$

Thus, the first character in the first row, corresponding to $x_0 = 1$ should be the only 0 in the table. As another example, the next-to-last character in column 63 of the eleventh row, corresponding to $x_0 = 10 \cdot 64 + 63 = 703$, should be an asterisk, as x_{170} is the first occurrence of the number 1 in this sequence, and the first row in your solution should be the following:

```
01111111211112211221111121722271211222317121117122211772222227 71
```

In particular, the third character in this row is a 1, since when $x_0 = 3$, $x_7 = 1$, as described above.

8 Determinants

A *square matrix* is an $n \times n$ grid of numbers for some integer n. We write a_{ij} for the entry in the jth column of the ith row of the matrix A; thus, we write:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

The *determinant* of a square matrix A, written as

$$|A| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix}$$

is defined as follows:

- If A is a 1×1 matrix, $|A| = |a_{11}|$ is the lone entry a_{11}.
- Otherwise, if A is an $n \times n$ matrix with $n > 1$, define $|A|$ recursively as follows:
  - Let Bi be the $(n-1) \times (n-1)$ matrix obtained by crossing off the first row and the ith column. That is,

$$B_i = \begin{bmatrix} a_{21} & a_{22} & \cdots & a_{2,i-1} & a_{2,i+1} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3,i-1} & a_{3,i+1} & \cdots & a_{3n} \\ \vdots & \vdots & \ddots & \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{n,i-1} & a_{n,i+1} & \cdots & a_{nn} \end{bmatrix}$$

  - Then $|A| = a_{11}|B_1| - a_{12}|B_2| + \cdots + (-1)^n|B_n|$.

For example,

$$\begin{vmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 1 & -1 & 2 \end{vmatrix} = 0 \cdot \begin{vmatrix} 4 & 5 \\ -1 & 2 \end{vmatrix} - 1 \cdot \begin{vmatrix} 3 & 5 \\ 1 & 2 \end{vmatrix} + 2 \cdot \begin{vmatrix} 3 & 4 \\ 1 & -1 \end{vmatrix}$$

Working out each of these 2×2 determinants and remembering that the determinant of a 1×1 matrix is the entry itself, we find

$$|A| = 0 \cdot (4 \cdot |2| - 5 \cdot |-1|) - 1 \cdot (3 \cdot |2| - 5 \cdot |1|) + 2 \cdot (3 \cdot |-1| - 4 \cdot |1|) = 0 \cdot (8+5) - 1 \cdot (6-5) + 2 \cdot (-3-4) = -15.$$

Write a program which takes as input the number n with $n \leq 8$ on a line by itself, followed by an $n \times n$ grid of integers, each between -100 and 100. Your program should then print the determinant of the resulting matrix. For example, if the input is

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

the output should be `-15`, as computed above.