2005 Quinnipiac University Programming Competition for High School Students

Quinnipiac University

April 9, 2005

Submit problems via the URL

`http://cs-scorpion.quinnipiac.edu/upload.html`

Fill in your team number and the problem number, and browse for the correct file.

1 Adding consecutive numbers

Write a program which takes two integers on separate lines as input and prints the sum of all the integers between them, including the two given numbers. Note that the numbers can appear in either order. You may assume that both numbers are between $-10,000$ and $10,000$. For example, if the input is as follows:

```
10
4
```

the output should be 49, since $10 + 9 + 8 + 7 + 6 + 5 + 4 = 49$. Similarly, the input

```
-3
10
```

should produce the output 49, and if the input is

```
2005
2005
```

the expected output is 2005.

2 Alphabet Snake

Write a program which takes two numbers m and n as input, each on a line by itself, and prints an $m \times n$ grid of uppercase letters (m rows and n columns) according to the following rules.

1. The first row contains consecutive letters *from left to right*, beginning with an 'A'.
2. The next row contains consecutive letters *from right to left*, beginning where the previous row left off.
3. In subsequent rows, the direction alternates. In an odd row, letters are printed from left to right; in an even row, they are printed from right to left.
4. Whenever a 'Z' is printed, the next letter is an 'A'.

For example, if the input is

```
11
8
```

the output should be

```
ABCDEFGHIJK
VUTSRQPONML
WXYZABCDEFG
RQPONMLKJIH
STUVWXYZABC
NMLKJIHGFED
OPQRSTUVWXY
JIHGFEDCBAZ
```

3 Rearranging letters in a sentence

In this problem we take the words of a sentence and rearrange the letters within each word. The first and last letters of each word are unchanged, and the letters in the rest of the word are reversed. For example, the words “Quinnipiac University” become “Qaipinniuc Utisreviny” under this transformation.

The input for this problem consists of the number of words in the sentence on a line by itself, followed by each word, along with any punctuation. All punctuation will consist of a single character at the end of a word. The punctuation should be ignored and the word should be transformed as described above. The output should consist of the transformed words in order along with the attached punctuation, with one space between each word. For example, if the input is

```
8
Today,
I
am
writing
lots
of
fun
programs!
```

the output should be

```
Tadoy, I am wnitirg ltos of fun pmargors!
```

4 Bowling scores

A bowling game consists of ten frames. In each frame, a player gets two balls to knock down ten pins. If a player knocks down all ten pins with the first ball, that is called a *strike*; if pins are left standing after the first ball, but the player knocks down the remaining pins with the second ball, that is a *spare*; a frame in which if pins are left standing after both attempts is called an *open frame*.

The scoring works as follows: in an open frame, the player is credited with the number of pins knocked down and is given that many points for the frame. For a spare, a player is given ten points for the frame *plus* the number of pins knocked down by the first ball in the following frame. For a strike, a player is given ten points for the frame *plus* the number of pins knocked down by the next *two* balls rolled. If a player rolls a strike or a spare in the tenth frame, that player then bowls an eleventh (or possible even a twelfth) frame to determine the number of points added to the tenth frame.

For example, suppose the player knocks down pins in each frame according to the following table:

Frame	**Ball 1+Ball 2**	**Type of Frame**	**Score**
1	3 + 6	Open	9
2	10 + NA	Strike	$10 + 10 + 10 = 30$
3	10 + NA	Strike	$10 + 10 + 2 = 22$
4	10 + NA	Strike	$10 + 2 + 7 = 19$
5	2 + 7	Open	9
6	0 + 10	Spare	$10 + 8 = 18$
7	8 + 2	Spare	$10 + 0 = 10$
8	0 + 5	Open	5
9	8 + 2	Spare	$10 + 3 = 13$
10	3 + 7	Spare	$10 + 6 = 16$
11	6 + NA	Unfinished	0
Total			**151**

Note that the strike in the fourth frame is counted as the third ball in the second frame, the second ball in the third frame, and as the first ball in the fourth frame. Also note that the eleventh frame does not count; it is only bowled to add points to the tenth frame, so there is no need to finish it.

Write a program which takes as input a single line in which each character represents the number of pins knocked down by a particular ball and computes the score for that game. The digits `'0'` through `'9'` represent the corresponding number of pins, and a lowercase `'x'` represents a ball that knocks down ten pins. You may assume that the input is properly formed, and it has no spaces and the correct number of characters. Your program should print out the score. For example, the input corresponding to the example above would be

```
36xxx270x820582376
```

and the output in this case should be 148. A perfect score of 300 would be the result of the following input.

```
xxxxxxxxxxxx
```

5 Avoiding substrings

A *bitstring of length* n is a sequence of n zeroes and ones. For example, 11001 is a bitstring of length 5. Given two bitstrings whose length is the same, we compare them by looking at the first (leftmost) position in which they are different, and say the bitstring with a zero is that position is *smaller* than the bitstring with a one in the corresponding position. For example, the bitstring 10111 is smaller than the bitstring 11001. Given a fixed length n and a collection of bitstrings whose lengths are less than or equal to n, we want to find the smallest bitstring of length n which does not contain any of the other bitstrings as substrings (or to state that no such bitstring exists).

For example, suppose we are looking for the smallest bitstring of length 6 which does not contain either 000 or 10 as a substring. We cannot have three consecutive zeroes, but we can begin with two zeroes followed by a one. Since a one cannot be followed by a zero, we fill the rest of the bitstring with ones. Therefore, the smallest possible bitstring is 001111.

The input for this problem is the length n on a line by itself, then another number specifying how many bitstrings are in the collection to be avoided, also on a line by itself, followed by each bitstring that must be avoided. You may assume that $n \leq 10$ and that there are at most 10 bitstrings to avoid. The output is either the smallest bitstring of length n which does not contain the other bitstrings as substrings, or the word "none" if there is no such bitstring. Here are several examples of input along with the corresponding output.

Input	6 2 000 10	4 2 1111 0	8 0	10 3 0000 1000 10010
Output	001111	none	00000000	0001001100

The first example is the one given above. In the second example, the only bitstring of length 4 that does not contain a zero is the bitstring 1111, but we also are avoiding this string, so there is nothing left. In the third example, there is nothing to avoid, so we can use the smallest bitstring of length 8, which is eight consecutive zeroes.

Finally, in the last example, we cannot begin with four zeroes, so we try starting with three. These must be followed by a one, but the one cannot be followed by three zeroes, so we append two zeroes and a one. Since the last four characters are currently 1001, we cannot append a zero, so we add a one, but now we can fill the bitstring with zeroes.

6 City Streets

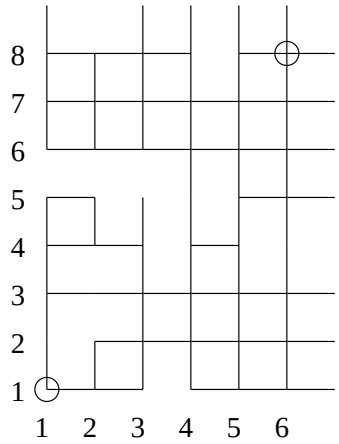
In the city of Quinnip, avenues run East/West and are numbered beginning with First Avenue along the southern edge of the city, and streets run North/South and are numbered beginning with First Street on the western edge of the city. The city is doing a significant amount of road work which makes portions of some roads impassible. Bob Cat lives at the corner of First Avenue and First Street. Every morning, he downloads the locations where the roads are impassible that day before taking a walk to visit one of his friends, all of whom live within 25 blocks. Of course, he wants to use a route as short as possible, walking only to the north or east, and at any intersection where he has a choice, he prefers to go north instead of east to avoid looking at the Sun. He wants you to write a program to help him figure out his route.

The input consists of two numbers, each on a line by itself, specifying (in order) the avenue and the street where Bob wants to go today, followed by a grid specifying the directions one can walk from each intersection. The characters in this grid have the following meaning:

Character	**Possible directions**
`X`	Neither north nor east is possible
`N`	Going north is possible; going east is not
`E`	Going east is possible; going north is not
`B`	Both going north and east are possible

Your program should print a string of `N`'s and `E`'s giving the order in which Bob should travel the streets if such a solution is possible; otherwise, print the words "`No solution`". For example, if Bob is going to Eighth Avenue and Sixth Street, he might enter the input given on the left below, which corresponds to the map on the right. The map includes the roads going to Ninth Avenue and Seventh Street.

```
8
6
BEBNBB
BBBBBB
BBBBBB
EXXNBB
BBNBNN
BEBBBB
NEBBBB
BBNBBB
```



In this case, the output of the program should be `NNEEENNNNENE`.

You may assume that the input contains no spaces, and no extra rows or columns; for example, if Bob is going to Eighth Avenue, the grid contains eight rows corresponding to the possible directions from each intersection from First Avenue to Eighth Avenue. Furthermore, his friend may live on First Avenue or on First Street, but you may assume that he does not visit someone else at the intersection of First Avenue and First Street.