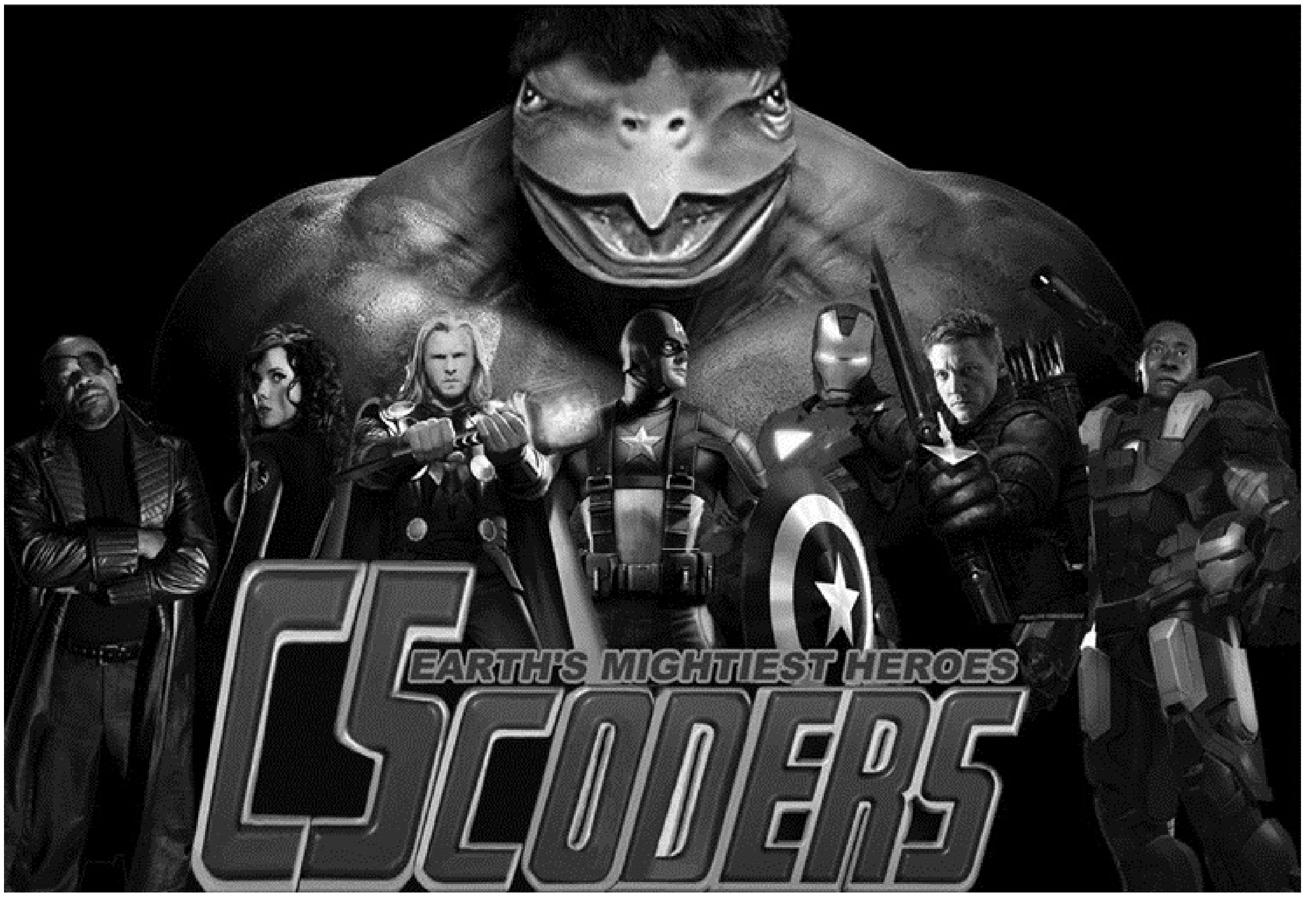
EARTH'S MIGHTIEST HEROES
CS CODERS

Practice 1 - Sum of Pairs

The Avengers are looking for ways to improve their math skills, to better outsmart their many opponents. Captain America suggests the following problem. Given a number n between 1 and 12, find all the pairs of numbers between 1 and 12 that when summed produces n. For instance, given the number 5, two possible pairs of numbers are 1,4 and 2,3. Note that the two numbers in a pair must be different. I.e., 3,3 is not a legal pair.

To make the problem more challenging, the pairs must be presented in *lexicographic* order. I.e., two pairs of numbers are compared by their lowest number. For instance, 1,5 would come before 2,4 when sorted in lexicographic order.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that, each line contains one test case: the number n.

Output: For each test case, you are to output in lexicographic order the different pairs of numbers between 1 and 12 that when summed produce n. If there are no such numbers, don't print anything.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
    private static ArrayList<Integer> solveSumPairs(int target)
```

which should return an ArrayList of Integers representing the solution. In the ArrayList returned, The 1st & 2nd numbers represent the first pair (lower number first), the 3rd & 4th numbers represent the second pair, etc...

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; that blank line should not be present in the actual output.

`Input:`	`Output:`
`4`	
`2`	`Pairs for 2:`
`3`	`Pairs for 3:  1 2`
`4`	`Pairs for 4:  1 3`
`5`	`Pairs for 5:  1 4, 2 3`

Practice 2 - Alphabet Distances

The Avengers are looking for ways to analyze secret codes, to found information about their many opponents. Thor suggests the following problem. Given two words of equal length, find the alphabet distance between matching letters in each word.

The alphabet distance between 2 letters x and y is defined by assigning ’A’=1, ’B’=2, ... ’Z’=26. Then the distance is $y - x$ if $y \geq x$, and $(y + 26) - x$ if $y < x$.

For instance, the alphabet distance between ’B’ and ’D’ is $4 - 2 = 2$, while the distance between ’D’ and ’B’ is $(2 + 26) - 4 = 24$.

You can think of the distance between x and y as the number of times x must be bumped up to reach y, with ’Z’ becoming ’A’ when bumped.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). After that, each line contains one test case with two words. You can assume that the words are between length 4 and 20, and in all capital letters (from A to Z).

Output: For each test case, you are to output the alphabet distances between the letters in order.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
    private static ArrayList<Integer> solveAlphabetDistance(String s1, String s2)
```

which should return an ArrayList of Integers that contains the distances in order.

Examples:

The output is shown with an extra blank line so that each test case input is aligned with the output; that blank line should not be present in the actual output.

Input:	Output:
5	
AAAA ABCD	Distances: 0 1 2 3
ABCD AAAA	Distances: 0 25 24 23
DARK LOKI	Distances: 8 14 19 24
STRONG THANOS	Distances: 1 14 9 25 1 12
DEADLY ULTIMO	Distances: 17 7 19 5 1 16

Practice 3 - String Transformations

The Avengers are looking for ways to encrypt their messages, to hide information from their many opponents. Iron Man suggests the following approach.

Given several string transformation rules, transform a target string. Each string transformation rule (production) transforms a single letter into two letters, and is applied to the first occurrence of the letter in the target string. If the letter does not appear then the string is unchanged. Once a new string is produced, it is used as the target string of the next transformation rule, until no rules are left.

For instance, the production $A \rightarrow OA$ applied to the string ”CAT” would yield the string ”COAT”. Applied to the string ”DATA” it would yield the string ”DOATA”.

Input/Output Format:

Input: The first line in the test data file contains the number of test cases (< 100). Each test case consists of two lines. The first line contains a number of transformation rules as pairs of words *lhsrhs* where *lhs* is a single letter, and *rhs* is two letters. The second line contains the list of words to be transformed. You may assume all words are in capital letters.

Output: For each test case, you are to output the transformed words in their original order, separated by spaces on a single line.

Note: We have provided a skeleton program that reads the input and prints the output. All you need to do is provide the body of the following procedure:

```
    private static String solveStringTransform(
           ArrayList<String> lhs, ArrayList<String> rhs, String target)
```

where given a set of production rules in two ArrayLists and a target string, returns the transformed string.

Examples:

The output is shown with some extra blank lines so that each test case input is aligned with the output; these blank lines should not be present in the actual output.

Input:	Output:
3	
A OO	
A BAT ART DATA	Transformed strings: OO BOOT OORT DOOTA
A AR A AO A TH	
A VS HULK	Transformed strings: THOR VS HULK
X HX X UX X LK	
X JOINS AVENGERS	Transformed strings: HULK JOINS AVENGERS