



# **ECOO 2016**

## **Programming Contest**

### **Questions**

Regional Competition (Round 2)

April 30, 2016

**Sheridan** | Get  
Creative

Questions made possible in part through the support of the  
Sheridan College Faculty of Applied Science and Technology.

# Problem 1: Palindrome Panic

---

A palindrome is a word that has the exact same sequence of letters whether you read it left to right or right to left. The words “noon” and “radar” are palindromes. The words “noob” and “palindrome” are not.

Patty loves palindromes. She finds them soothing. Patty knows that you can turn any word into a palindrome by adding letters to the left and/or the right. For example, you can turn “ECOO” into “OOCECOO” or “ECOOCE” or even “DOOCECOOD”. Patty loves palindromes so much that any time she sees a word written down somewhere, she immediately converts it to a palindrome in her head.

DATA11.txt (DATA12.txt for the second try) will contain 10 test cases. Each test case consists of a single line containing a string of lowercase letters of maximum length 1 000 000. Your code should output the minimum number of letters Patty would need to add to the left and/or the right in order to convert the string to a palindrome.

Note that the sample input below contains only 3 test cases but the data files will contain 10.

## *Sample Input*

```
ecoo  
impala  
anagram
```

## *Sample Output*

```
2  
3  
4
```

### **Question Development Team**

Sam Scott (Sheridan College)

Kevin Forest (Sheridan College)

Stella Lau (University of Cambridge)

Reyno Tilikaynen (University of Waterloo)

John Ketelaars (ECOO-CS Communications)

David Stermole (ECOO-CS President)

## Problem 2: CC+

---

CC+ is a cyclical cypher. A cyclical cypher is a method of encrypting messages in which every letter in a message is “rotated” some number of positions. For example, if the letters are rotated 1 position then 'a' becomes 'b', 'b' becomes 'c', and so on ('z' becomes 'a').

In a CC+ encryption (Cyclical Cypher Plus) the number of positions a given letter is rotated is based on an integer key value plus the sum of the letters to the right (where a = 0, b = 1, etc.). Messages to be encrypted by CC+ must only contain lowercase letters and space characters. No uppercase, digits, punctuation or other types of characters are allowed. Spaces in the original message are removed in the encrypted version.

Before the letters get rotated, special encoding characters are added to the left of the message string. The first two characters represent the number of words in the message as a base 26 number, using lowercase letters (a = 0, b = 1, etc.). For example, if there are 5 words, then these letters are 'af', which converts to 05. If there are 28 words, then the letters are 'bc', which converts to 12, the base-26 equivalent of the decimal number 28. This is followed by a string of characters, each of which represents the length of one of the words in the message (in the order they appeared).

Here’s an example encoding using key 5:

Message:	put the lime in the coconut
Step 1:	agddecdh put the lime in the coconut
Step 2:	bbvspljgzkqxextiaokcpwpljvtfsy

DATA21.txt (DATA22.txt for the second try) will contain 10 test cases. Each test case will consist of two lines. The first line contains an integer  $K$  ( $1 \leq K \leq 2\,000\,000\,000$ ) and the second contains an encrypted or unencrypted message. Each message contains between 2 and 675 words, each from 1 to 25 characters in length. Words in unencrypted messages are separated by a single space character. Your program should either encode or decode each message (as appropriate) using key  $K$  and output the result on a single line.

Note that the sample input below consists of only two test cases, but the data files will contain 10.

### *Sample Input*

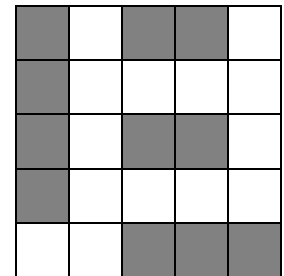
```
5
put the lime in the coconut
13
ccxupnkiivspyqdtlsshwc
```

### *Sample Output*

```
1. bbvspljgzkqxextiaokcpwpljvtfsy
2. and drink it all up
```

# Problem 3: BattleShip

In a popular mobile app, two friends can play a game of battleship against one another. In this version of battleship, each player has a grid of squares in which they can place their ships vertically or horizontally. Each ship takes up a vertical or horizontal row of grid squares equal to its length. For example, on the board shown at right, the player has placed two ships of length 2 and one each of lengths 3 and 4.



The rules of the app state that you are not allowed to place ships adjacent to one another. This means that two squares occupied by different ships must not share a side or a corner.

h				
h				
	m	h	h	
	m			

The players can't see their opponent's arrangement of ships. They take turns to fire torpedoes blindly into their opponent's grid. Each torpedo hits one grid square. If there is a ship covering that square, it's a hit. If not, it's a miss. The game is over when one player has hit every grid square covered by an opponent's ship.

The board at left shows the other player's view of the board shown above, after 6 shots have been fired. Note that all this player knows is where she has scored hits (h) and misses (m).

In this version of the game, your opponent does not tell you when you have finished hitting a ship. In the example above, one of the length 2 ships is finished but the player firing shots won't know she's finished with it until she fires a torpedo into the grid square to the right of it and misses.

DATA31.txt (DATA32.txt for the second try) will contain 10 test cases. The first line of each test case will contain two integers  $S$  and  $W$ , where  $S$  is the size of the board ( $1 \leq S \leq 100$ ) and  $W$  is the width of a boat ( $1 \leq W \leq S$ ). The next  $S$  lines will contain  $S$  characters each, representing the hits and misses to the opponent's ships, represented by lowercase h and m characters. All other grid squares will be filled with a . character (ASCII 46). You must output a single line containing an integer, representing the number of possible (and known) locations for a ship of width  $W$  given the hits and misses so far.

Note that the sample input below contains only 1 test case, but the data files will contain 10.

## Sample Input

```
5 3
.....
.hm..
.....
.....
.....
```

## Sample Output

```
14
```

# Problem 4: Hop, Skip and Jump




---

Hop, Skip and Jump are brothers seeking valuable items (the Cup, the Flag and the Treasure) in a 2D world of ladders and platforms. Their world is a grid of characters and each location on the grid is one of three possible types: a platform (the = character), a ladder (the # character) or clear (the . character). A brother can never occupy a location that has a platform in it, but he can occupy any location that is clear or contains a ladder. A brother can also stand in the location immediately above a platform or a ladder, as long as the location they are occupying is clear or contains a ladder.

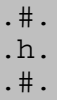
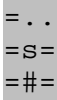
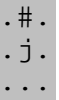
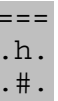
## Basic Movement Rules

All three brothers are affected by gravity. A brother is falling if he is currently on a clear location and the location beneath him is also clear. A falling brother always moves down until the location below him contains a platform or a ladder. A brother can fall an infinite distance without getting hurt. All three brothers can move one spot left or right, as long as they are not falling and as long as the location they are moving to does not contain a platform. All three brothers can climb up as long as their current location contains a ladder and the location above them does not contain a platform. All three brothers can climb down if the space they are occupying contains a ladder and the space below them does not contain a platform. All three brothers can also climb down if the space below them contains a ladder and they are not falling.

Here are some examples (h = Hop, s = Skip, j = Jump):

**a**  **b**  **c** 

In example a, Hop can move left or right. In example b, Skip can move right but not left. In example c, Jump can't move left or right because he is falling.

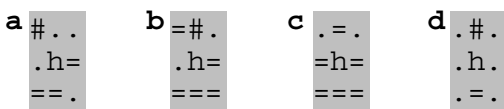
**d**  **e**  **f**  **g** 

In example d, if Hop's location contains a ladder, he can climb up or down, otherwise he can only climb down. In example e, if Skip's location contains a ladder, he can climb up or down, otherwise he can only climb down. In example f, if Jump's location contains a ladder, he can climb up or down, otherwise he is falling. In example g, Hop can only climb down regardless of whether or not his location contains a ladder.

## Special Moves for Hop

As long as he isn't falling, Hop can hop up to the left or right. A hop moves Hop to a target location one spot sideways (left or right) and one spot up, provided the location immediately above Hop and the target location are both not a platform. Hop can also hop straight up (as long as he's not falling) provided the location above him isn't occupied by a platform.

Here are some examples:

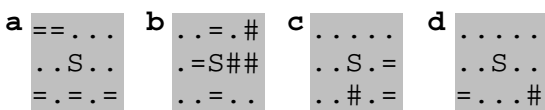


In example a, Hop can hop up, left and right. In example b, Hop can hop up and right but not left. In example c, Hop cannot hop in any direction. In example d, Hop can hop up, left and right.

### Special Moves for Skip

As long as he isn't falling, Skip can skip to the left or right. A skip moves Skip to a target location two spots sideways (left or right), provided the target location and the intervening location are both not occupied by a platform.

Here are some examples:

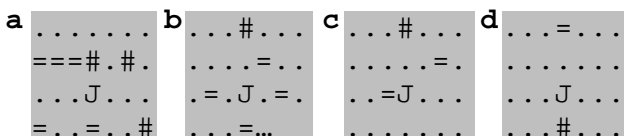


In example a, Skip can skip left or right. In example b, Skip can skip right but not left. In example c, Skip can skip left but not right. In example d, if Skip's location contains a ladder, he can skip left and right, otherwise he is falling.

### Special Moves for Jump

Jump can Hop and Skip just like his brothers. As long as he isn't falling, Jump can also jump to the left or right using a long jump or a high jump and he can jump up with a spring jump. A long jump moves Jump to a target location three spots sideways (left or right), provided the target location and the intervening locations are all not occupied by platforms. A high jump moves Jump to a target location that is two spots sideways (left or right) and one spot up, provided the location beside Jump (in the direction of his jump), the location above Jump, the location above the location beside Jump, and the target location are all not occupied by platforms. Finally a spring jump moves Jump two spots up, provided the target location and the intervening location are both not occupied by a platform.

Here are some examples:



In example a, Jump can long jump left and right, can high jump right but not left, and can spring jump. In example b, Jump can't long jump at all and can't high jump right, but can high jump left or spring jump. In example c, if Jump's location contains a ladder, he can long jump right and can spring jump but he can't long jump left or high jump in either direction. If Jump's location is clear in example c, then he's falling. In example d, Jump can long jump and high jump in both directions, but cannot spring jump.

DATA41.txt (DATA42.txt for the second try) will contain 10 test cases. The first line of each test case consists of two integers  $W$  and  $H$  ( $0 < W, H < 200$ ) representing the width and height of the level. The next  $H$  lines of data will consist of  $W$  characters, representing the layout of the level. The three brother locations will be represented by lowercase  $h$ ,  $s$  and  $j$ . Initially, the brothers always occupy a location that is clear. The three goal locations will be represented by an uppercase  $C$ ,  $F$  and  $T$  (for Cup, Flag and Treasure). The remaining locations will each be one of the following: a  $.$  character (ASCII 46) for clear, a  $\#$  character (ASCII 35) for a ladder or an  $=$  character (ASCII 61) for a platform.

For each test case you must output three lines using only capital letters. Each line should consist of the name of a brother, followed by a space, followed by the first letter of each goal that brother is able to reach from their start position using their movement rules. The output for each test case should show Hop first, then Skip, then Jump. The goals each brother can reach should be listed in alphabetical order ( $C$ , then  $F$ , then  $T$ ). Note that each goal that a brother can reach could be reachable via a completely different path from the others. If a brother cannot reach any of the goals print only that brother's name. You will be allowed 45 seconds of total execution time for all 10 test cases.

Note that the sample input below contains only 1 test case but the data files will contain 10. For readability, you can display your output in groups of 3 lines separated by a blank line.

### *Sample Input*

```
10 10
#.....#.
#=#...#..#.
#.=...C.#.
..#...===#.
#. #.....##
..#...=.##
.j...#...##
.=...s...#
=..h==T..F
...=.==...=
```

### *Sample Output*

```
HOP T
SKIP T
JUMP CFT
```