# C# Part 2 Exam – 14 September – Morning

## Problem 1 – Multiverse Communication

One day, after eating too much pizza, the **master-programmers** Niki, Toni and Ivo started talking about “highly-intelligent” topics – how the universe **started**, who created it, are there other **advanced** forms of life, what kind of girls are hot and so on. Suddenly they received an **answer** from beyond to one of the biggest mysteries for the mankind – there is more than one universe in the space-time continuum! As a matter of fact – they are **infinite** – a multiverse to rule them all! How cool is that, huh?

So, back to our story – somewhere in between the wormholes, dark matter and a lot of space-flying Zerg Mutalisks, there was another universe almost **identical** to ours. The very same day after eating too much spaghetti, Ikin, Inot and Ovi (being well trained Terran Ghosts), decided to send telepathically an **encrypted numerical message** to our well-known software engineers.

The sent message is made of the following digits:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | CHU | **3** | IVA | **6** | POQ | **9** | K-A | **12** | PLA |
| **1** | TEL | **4** | EMY | **7** | ERI | **10** | IIA |  |  |
| **2** | OFT | **5** | VNB | **8** | CAD | **11** | YLO |  |  |

### The message is written as a sequence of digits. The last digit of the number (the most right one) has a value as shown in the above table. The next digit on the left has a value 13 times bigger than the shown in the above table, the next digit on the left has 13\*13 times bigger value than the shown in the table and so on. Since our masters are too lazy after so much pizza and do not want to think (and code C# too), you task is to translate the message into its decimal representation. With your help, our heroes can fall asleep calmly, knowing other universes exist somewhere.

### Input

The input data consists of a single line – the message from the parallel universe.

The input data will always be valid and in the described format. There is no need to check it explicitly.

### Output

The output data consists of a single line holding the calculated decimal representation of the given message number and should be printed at the console.

### Constraints

* The input number will have between 1 and 9 digits.
* Allowed working time for your program: 0.1 seconds. Allowed memory: 64 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Explanation** |
| OFT | 2 | From the table **OFT** means **2** in 13th based numeral system. Message is **2**. After converting it to decimal – the answer is **2**. |
| IVAYLO | 50 | From the table **IVA** means **3** and **YLO** means **B (11)** in 13th based numeral system. Message is **3B**. After converting it to decimal – the answer is **50**. |
| TELERIK-ACADEMY | 45569 | From the table **TEL** means **1**, **ERI** means **7**, **K-A** means **9**, **CAD** means **8** and **EMY** means **4** in 13th based numeral system. Message is **17984**. After converting it to decimal – the answer is **45569**. |

## Problem 2 – Magic Words

We are given a sequence of **n** magic words. The words are magic because they are capable to perform two “magic” operations: “**reorder**” and “**print**”.

**Reorder**: The reordering operation over **n** words is performed by sequentially moving the words at positions 0, 1, …, n-1 to positions corresponding to their lengths. More precisely, first the word **w0** from position **0** is moved to position **len(w0) % (n+1)**, then the word **w1** from position **1** is moved to position **len(w1) % (n+1)**, and so on. Finally the word **wn-1** from position **n-1** is moved to position **len(wn-1) % (n+1)**. Note that positions are numbered from **0** to **n** and position **0** is just before the leftmost word and position **n** is just after the rightmost word.

For example, if we have **n = 3** words {"**hi**", "**academy**", "**exam**"}, they will be reordered **3** times this way:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Position* | ***0*** | *1* | *2* | 0 🡪 2 | *Position* | *0* | ***1*** | *2* | 1 🡪 2 |
| *Word* | **hi** | academy | exam | *Word* | academy | **hi** | exam |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Position* | *0* | *1* | ***2*** | 2 🡪 0 | *Position* | *0* | *1* | *2* |  |
| *Word* | academy | hi | **exam** | *Word* | exam | academy | hi |

**Print:** The printing process is simple. It just first prints the first letter of all words, then the second letter of all words (when it exists), then the third letter of all words (when exists), etc. As a result the printing obtains a sequence of letters from the input words.

Your task is to write a program that reads **n** words, performs the magic operations “reorder” and “print” over them and outputs the obtained sequence of letters.

**Input**

The input data should be read from the console. The first line holds a single integer number **n**. At each of the next **n** lines there is a single word.

**Output**

The output data consists of a single text line holding the obtained result.

### Constraints

* The number of words **n** will be in the range [1...1000].
* Each word will have between 0 and 1000 English letters, each in the range [a…z].
* Allowed working time for your program: 0.06 seconds. Allowed memory: 32 MB.

### Examples

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 3  hi  academy  exam | eahxciaamdemy | 2  you  win | wyionu | 1  hi | hi | 4  nakov  wrote  this  problem | wnptrarhokoitobsevlem |

## Problem 3 – Tron 3D

### You are implementing a 3D version of the game Tron. If you are not familiar with the standard 2D version, it's a simple reaction game where players race on a grid, leaving colored trails behind them; any player who touches the trails is eliminated, and the last player alive is the winner.

### In the example, the two players start at points A and B respectively, moving on the grid and turning left and right, trying to surround each other. In point C, the red player crashes into the blue player's trail and loses the game.

### Tron 3D is similar, with the players racing on the surface of a 3D cube instead of inside a rectangle.

### There are two players, starting from the centers of two opposing walls of the cube, and turned in the same direction (towards one another). They move on a grid on the surface, and in every game cycle, they move one position in their current direction; before every move, they can turn left or right (turning is instant and doesn’t count as taking a cycle of game time).

### In the example, the two players start at the centers of the front and back faces; they race on several walls of the cube, until the red player surrounds the blue player, and the blue player crashes in point D.

### Additional rules for movement:

* two of the walls of the cube are **forbidden** – а player who tries to move on one of these walls crashes and loses the game; the forbidden walls are opposite one another (on the picture they are marked with diagonal lines);
* when a player reaches an edge of the cube, he continues moving on the next wall, in the same direction (see point A on the example);
* a player can move on the edge of the cube (see segment B-C); when he reaches a corner of the cube, he must turn left or right (see point C).

### The game ends when one or both of the players crash. If both players crash on the same game cycle, the game is a draw; otherwise, the one who didn't crash wins. Your program will read a sequence of moves from the console, and determine the winner and the distance between the start and endpoint of the red player, along the grid (in this case, 8 – 4 down, and 4 along the bottom edge).

### Input

The input data should be read from the console. On the first line, you will read three integers - X, Y and Z - representing the dimensions of the cube. X and Y represent the dimensions of the walls on which the players start, Y and Z are the dimensions of the forbidden walls, and X and Z are the dimensions of the other two walls. The players start in the center of the two opposite X\*Y walls, and move in the direction of edge X (towards each other; see example input 2 below).

On the second and third line you will read two strings of characters representing the motion of the red and blue players respectively. The motion is represented as a string of M, L and R characters, where M means "move without turning, L means "turn left", and R means "turn right".

The input data will be correct and there is no need to check it explicitly.

### Output

The output data should be printed on the console. On the first output line you should print “RED”, “BLUE” or “DRAW”, depending on who won the game.

On the second line, print the distance between the start and end points of the red player, measured along the playing grid, as an integer (if the red player crashes into a forbidden wall, his final position is the point where he crashed).

### Constraints

* The numbers **X**, **Y** and **Z** are positive even integers in the range [2…50].
* The motion strings will be between 2 and 120 characters long (only characters M, L and R).
* The length of the motion strings will be long enough to finish the game.
* Allowed working time for your program: 0.1 seconds. Allowed memory: 64 MB.

### Examples

|  |  |
| --- | --- |
| **Example input (example on picture)** | **Example output** |
| 8 4 6  MMLMMMMRMRMMLMMRMMRMLMMRMMRMLMMLMMMLMMM  LMMMMRMRMMMLMMRMMMMLMLMMMMRMLMMRMMMMRMM | RED  8 |

|  |  |
| --- | --- |
| **Example input (players move without turning and crash into each other)** | **Example output** |
| 4 2 4  MMMM  MMMM | DRAW  3 |

## Problem 4 – Decode and Decrypt

So, being a KO-NE (Key Observation – Notification Expert), you really don't like the idea, that half the company you work for has started using a new method of messaging. This new method encrypts and encodes (compresses) the messages. Encoding is all well and good – company's saving on Broadband and all that jazz – but it's the encryption part you really have a problem with. After all, your job is "observation" and you really can't be effective at that when you can't even read the damn thing.

Good thing is, being as good as you are, you found the source of the idea for the messaging system – some old article, describing a primitive but effective encoding and encryption algorithm. So much for security by obscurity. Here's the encryption and encoding algorithm description from the article:

* We are given a **message** and a **cypher**
  + The message is the text the user wants to transmit
  + The cypher is a string which is used to encrypt and decrypt the message
  + The encrypted message is called **cypherText**
* We define a function **Encrypt**, which takes a message and a cypher:
  + It iterates over the symbols in the message and the cypher
  + For each **pair of symbols**, it takes their **codes** (in the table below) and computes the **bitwise XOR** of the **symbol in the message** with the **symbol in the cypher**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

* + The **resulting code** is **summed** with the **ASCII code of the character 'A'** (65), giving a new ASCII code
  + The **character corresponding to this new ASCII code** is the **encrypted representation** of the respective **character in the message**
  + If the cypher string is shorter than the message, using it symbols loops to the beginning of the cypher. E.g. for a message "**ABCDE**" and a cypher "**PQR**" we will have:
    - **'A' encrypted with 'P'** = 'P', **'B' encrypted with 'Q'** = 'R', **'C' encrypted with 'R'** = 'T', **'D' encrypted with 'P'** = 'M', **'E' encrypted with 'Q'** = 'U'
  + If the message string is shorter than the cypher, some of its symbols will be encoded several times, until all of the cypher symbols are used.
    - E.g. for a message "**ABC**" and a cypher "**PQRST**", we will have:
    - **'A' encrypted with 'P'** and **the result** ('P') **encrypted with 'S'** = '^' (ASCII 94),
    - **'B' encrypted with 'Q'** and **the result** ('R') **encrypted with 'T'** = 'C',
    - **'C' encrypted only with 'R'** = 'T'
* We define a function **Encode**, which takes a string of text to compress:
  + It looks for sequences of symbols which are the same(e.g. 'aaaaa')
  + For each sequence of same symbols, the function replaces the sequence with a number representing the count of repeated symbols, followed immediately by the symbol which is repeated. This is called run-length encoding. E.g. **for the text "aaaabbbccccaa" we will have "4a3b4caa"**.
    - The function **replaces symbols** in the aforementioned way **ONLY** **if the run-length encoding** of the **same-symbol sequence** is **shorter than the sequence itself**
    - That's why in the example above the last two a's remain the same – '2a' has the same length as 'aa'
* Given the two functions, and given a message and a cypher, the encrypted message should be:
  + **Encode(Encrypt(message, cypher) + cypher) + lengthOfCypher**
    - '+' denotes concatenation, the two functions return strings and 'lengthOfCypher' is a number, which is equal to the number of symbols in the cypher
    - i.e. the message is encrypted with the cypher, then the cipher is added for decrypting purposes, then the result is compressed and a number denoting the length of the cypher before compression is added to the compressed string

Now, since you are very good, you know that the described **Encrypt function actually works both ways** – i.e. if something was encrypted with the function and a cypher, calling Encrypt again with the same cypher, but with the encrypted text, you will receive the original text.

* For example, **Encrypt("ABCDE", "PQR") = "PRTMU"** and **Encrypt("PRTMU", "PQR") = "ABCDE"**

|  |  |
| --- | --- |
| **Input example** | **Output example** |
| BKOXHI\EQOGX[YSOFTWARE8 | TELERIKACADEMY |
| ABBAA6BA7 | AAABB |
| KKICXDE3P5 | JOHNY |
| Constraints  * All symbols in the message will have ASCII codes in the range [65; 127] * The original message for any encrypted message will always contain only capital English letters * The original message will be no longer than 1500 symbols * Allowed working time for your program: 0.1 seconds. Allowed memory: 64 MB. | |

* + Where the fist parameter of Encrypt is the message and the second is the cypher

The Encode function is also relatively easy to reverse – just take the numbers and print the symbol after each number the corresponding … number … of times.

Now all you have to do is put the pieces together and you can once again spy on the messaging in the company.

Write a program, which by **given an encrypted** (with the above described method) **message**, **recovers the original message**

### Input

The input data should be read from the console.

On the only input line there will be the cyphered message

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output data should be printed on the console. Print exactly one line – the original message (decrypted and decoded)

## Problem 5 – Featuring with Grisko

Grisko is a famous Bulgarian singer. As his personal website says “*Grisko is the author of almost all modern Bulgarian hits*” (yeah, sure). Anyway, Grisko has a requirement if you want to do a featuring with him you should only use **words with** **no two consecutive equal characters** in your rhymes. Let’s imagine that someone somewhere wants to sing along with Grisko and let’s imagine that you want to help that poor guy by writing a program.

You are given few letters. Write a program that finds **the number of all words with no two consecutive equal character that can be generated by reordering the given letters**. The generated words should contain all given letters. If the given word meets the requirements it should also be considered in the count.

### Input

The input data should be read from the console.

On the only input line there will be a single word containing all the letters that you should use for generating the words.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

The output data should be printed on the console.

On the only output line write the number of words found.

### Constraints

* The number of the given letters will be between 1 and 10, inclusive.
* All given letters will be small Latin letters (‘a’ – ‘z’)
* Allowed working time for your program: 0.35 seconds. Allowed memory: 64 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Example input** | **Example output** | **Explanation** |
| xy | 2 | Two possible words: “xy” and “yx” |
| xxxy | 0 | It is impossible to construct a word with these letters. |
| aahhhaa | 1 | The only possible word is “ahahaha”. |
| nopqrstuvw | 3628800 | There are 3628800 possible words. |

**Good luck!**