# C# Part 2 Exam – 14 September 2013 – Evening

## Problem 1 – Zerg!!!

The Prelate **Zeratul** and the High Templar **Tassadar**, highly psionic alien life forms from the Protoss race, got together to understand the low-level basic **xenomorphic** Zerg (in Bulgarian “дзверг“) communication. After years of energy **transformation**, deep meditation and thousands of Hydralisks cut into pieces, they finally found the answer – the awful creatures send each other **encrypted numeric messages** (like SMS but without mobile phones).

The messages are made of the following digits:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Rawr | **3** | Ssst | **6** | Mrrr | **9** | Uaha | **12** | Djav |
| **1** | Rrrr | **4** | Grrr | **7** | Psst | **10** | Zzzz | **13** | Myau |
| **2** | Hsst | **5** | Rarr | **8** | Uaah | **11** | Bauu | **14** | Gruh |

### Messages are 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 15 times bigger than the shown in the above table, the next digit on the left has 15\*15 times bigger value than the shown in the table and so on. Since our Protoss warriors are too advanced to calculate such easy expressions (and on Aiur they do not use C# too), you task is to translate the message into its decimal representation. With your help, our heroes can finally understand the Zerg logic and bring peace to the entire Koprulu sector.

### Input

The input data consists of a single line – the Zerg message.

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** |
| Rarr | 5 | From the table Rarr means **5** in 15th based numeral system. Message is **5**. After converting it to decimal – the answer is **5**. |
| HsstSsstSsst | 498 | From the table **Hsst** means **2**, **Ssst** means **3** and **Ssst** means **3** in 15th based numeral system. Message is **233**. After converting it to decimal – the answer is **498**. |
| GruhMyauDjav | 3357 | From the table **Gruh** means **(E)** **14**, **Myau** means **(D)** **13** and **Djav** means **(C)** **12** in 15th based numeral system. Message is **EDC**. After converting it to decimal – the answer is **3357**. |

## Problem 2 – Moving Letters

Nakov is a keen fan of cryptography. When he was a child, he invented an encryption algorithm called “**Moving Letters**”. It performs two steps: **extracts the letters** from the input message and **moves each letter** a few positions on the right according to its alphabetical order. Your task is to write a program that reads an input message, encrypts it through the “**Moving Letters**” algorithm and prints the result.

### Extracting the Letters

The input message is given as a sequence of words separated by a space. The words are converted into a **sequence of letters** the following way: until all words disappear, the last letter of each word (if exists) is removed from the word and is appended to the output sequence of letters.

### Moving the Letters

The next step in the encryption algorithm is to move each letter (from positions 0, 1, …, n-1) on the right **k** times. The number **k** is taken from the number of the letter in the Latin alphabet regardless of its casing ('a' 🡪 1, 'b' 🡪 2, …, 'z' 🡪 26). When a letter is moved to the right, if it is the last letter of the sequence, its next position is the first position in the sequence, just before all the letters.

### Example

Let's the input is "**Fun exam right**". It is first split into 3 words: {"**Fun**", "**exam**", "**right**"}. Then the last letters are extracted: {"**Fun**", "**exam**", "**right**"} 🡪 "**nmt**" 🡪 {"**Fu**", "**exa**", "**righ**"} 🡪 "**nmtuah**" 🡪 {"**F**", "**ex**", "**rig**"} 🡪 "**nmtuahFxg**" 🡪 {"**e**", "**ri**"} 🡪 "**nmtuahFxgei**" 🡪 {"**r**"} 🡪 "**nmtuahFxgeir**".

The moving of letters starts from the input sequence "**nmtuahFxgeir**" and sequentially moves right its first letter (at position 0), then its second letter (at position 1), and so on, and finally moves right its last letter (at position 11). First the letter '**n**' at position **0** is moved right **14** times: "**nmtuahFxgeir**" 🡪 "**mtnuahFxgeir**". Then the letter '**t**' at position **1** is moved **20** times right: "**mtnuahFxgeir**" 🡪 "**mnuahFxgetir**". Then the process continues: "**mnuahFxgetir**" 🡪 "**mnahFxgetiru**" 🡪 "**mnaFxgetiruh**" 🡪 "**mnaFxgetiruh**" 🡪 "**gmnaFxetiruh**" 🡪 "**gmnaFxtiruhe**" 🡪 "**gmnaiFxtruhe**" 🡪 "**gmrnaiFxtuhe**" 🡪 "**gmrnaiuFxthe**" 🡪 "**gmrnaihuFxte**" 🡪 "**gmrneaihuFxt**". The result is "**gmrneaihuFxt**".

### Input

The input data should be read from the console. It consists of a single line holding a sequence of words separated by a single space (followed by the "end of line" character).

### Output

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

### Constraints

* The input will be less than **256 KB** and will hold Latin letters separated by spaces.
* Allowed working time for your program: **0.35 seconds**. Allowed memory: **64 MB**.

### Sample Input and Output

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| Fun exam right | gmrneaihuFxt | Telerik Academy | AymlTiedkaerec | Hi exam | maiHex |

## Problem 3 – Trails 3D

### Trails is 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.

### Trails 3D is similar, with the players racing on the surface of a 3D parallelepiped 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, X and Z are the dimensions of the forbidden walls, and Y 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 Y (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". 'M' character may have a number prefix. This number prefix shows how many "move without turning" actions must be performed.

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  2MLM1MRM2MR2MLMLMR3MRM  LMMR2M4MRMLMRMR1M2MRM | BLUE  9 |

|  |  |
| --- | --- |
| **Example input (players move without turning and crash into each other)** | **Example output** |
| 4 2 4  3M1M  2M1M1M | DRAW  4 |

## Problem 4 – Encode and Encrypt

You are currently employed as a Junior MERIN-J (Management of Expression Research, Identification and Normalization – Job). One of your first tasks is to handle electronic messaging in your company, by providing a safe way to transfer messages. Not only should the messages be **encrypted** (made unreadable for anyone who does not know the cipher) in some way. The CEOs don't want to be wasting extra money on Broadband, so you also have to take care of **compressing (encoding)** the messages.

Since you don't have much time (about 6 hours or so), you need to think up of something quickly. Luckily, you have an old article on encryption and encoding, stating the following:

* 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 produced result 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

Write a program, **using the method above**, which encodes and encrypts a message with a cypher.

### Input

The input data should be read from the console.

On the **first line** of the input, there will be a single string, representing the **message**.

On the **second line** of the input, there will be a single string, representing the **cypher**.

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 cyphered message.

### Constraints

* All symbols in the message will be **capital English letters**
* Тhe message and the cypher will be no more than 1500 symbols each
* Allowed working time for your program: 0.1 seconds. Allowed memory: 64 MB.

### Examples

|  |  |
| --- | --- |
| **Input example** | **Output example** |
| TELERIKACADEMY  SOFTWARE | BKOXHI\EQOGX[YSOFTWARE8 |
| AAABB  BBBBBBA | ABBAA6BA7 |
| JOHNY  DEPPP | KKICXDE3P5 |

## Problem 5 – They are Green

Mani and Kerankata ate shkembe chorba while suddenly Ivo from Undercover showed on the **ultrasound**. He had **three eyes** and could eat without a spoon. He predicted the coming of the Zelenite (a pure green race, who loved to land on NASA headquarters). Mankata was there and gave the **green guys** a bottle of Brendy, a fine little fermented sault and bread whiskey. Kerancho popped some popcorn with his mind waves but suddenly a Turkish **series** aired on another TV channel. The boy was a walking dead guy with Bulgarian “**zabradka**”, kidnapped cutely because of his explosives. Now he is a prisoner without any hope, without any exit, with his body parts attached to **strange segments**. The background music was Koli Venova, Draifa and Grivna and he was not sure how much he could handle, because of it his legs started itching from the **inside of his bones**. His kidnappers were Moisei and his **semi-brother** Devil. The second looked like a bag of fertilizer on a Z-Factor show. Mani got brain implants making him dumb and shy. He even started liking “Where is Shaggy?”. Suddenly his tummy became **free zone** because the alien ship does not have **central** heating system, all of his fingers **transformed into a middle one**. If someone received the message, **please send an intergalactic backup!**

So… You are given **N** letters. Write a program that finds **the count of all words with no two consecutive equal characters 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 first input line there will be the number **N** – the count of the letters you should use. On each of the next **N** lines, there will be a single character.

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

### Output

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

### Constraints

* **N** 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.

|  |  |  |
| --- | --- | --- |
| **Example input** | **Example output** | **Explanation** |
| 4  a  b  a  a | 0 | It is impossible to construct a word with these letters. |
| 7  i  i  x  x  x  i  i | 1 | The only possible word is “ixixixi”. |

### Examples

|  |  |  |
| --- | --- | --- |
| **Example input** | **Example output** | **Explanation** |
| 2  b  a | 2 | Two possible words: “ab” and “ba” |
| 10  j  i  h  g  f  e  d  c  b  a | 3628800 | There are 3628800 possible words. |