## Problem 4 – Encrypt the Messages

You are working for a company which is very concerned about its information and communication. For this reason, they have invented an internal approach to communication between different departments – they are communicating to each other via **messages, which are reversed (written backwards) and then encrypted**. Your task is to write a program, which **encrypts** **all messages** in a specific communication, **prints them at the console as well as the total number of messages** that have been sent.

**At the beginning** of a communication, you will receive either the keyword “**START**” (upper case) or “**start**” (lower case), which indicates that you will **start receiving reversed and encrypted messages**. At the end of the communication, you will receive either the keyword “**END**” (upper case) or “**end**” (lower case), which indicates that the communication is over and you need to **show the encrypted messages’ content and total count**. Any **non-empty string** between the “start” and “end” keywords is considered a message. If **no messages have been sent** between the “**start**” and the “**end**” keywords, you should print on the console: **“No messages sent.”**

All messages are case-sensitive and consist of **letters**, **digits,** as well as **some special characters** – ‘’, ‘**,**’, ‘.’, ‘**?**’ and ‘**!**’. Letters **from A to M** are **converted** to letters **from N to Z** (A 🡪 N; B 🡪 O; … M 🡪 Z) and letters **from N to Z** are **converted** to letters **from A to M** (N 🡪 A; O 🡪 B; … Z 🡪 M). The **converted** letter should keep the **case** of the **original** letter. The **special characters** are converted in the following way: ‘’ (space) is converted to a **plus sign** (**‘ +’**), ‘**,**’ is converted to **‘%’**, ‘**.**’ is converted to **‘&’**, ‘**?**’ is converted to **‘#’** and ‘**!**’ is converted to **‘$’**. **Digits** (0-9) are **not converted** and stay the same.

For example, you receive the following message – “**Secret message 1!**” and you start encrypting it. Convert the 1st character ‘**!**’ to ‘**$**’, then the 2nd character – ‘**1**’ stays the same, then covert the 3rd character – ‘’ to ‘+’, ‘**e**’ 🡪 ’**r**’, ‘**g**’ 🡪 ‘**t**’, ‘**a**’ 🡪 ‘**n**’, ‘**s**’ 🡪 ‘**f**’, ‘**s**’ 🡪 ‘**f**’, ‘**e**’ 🡪 ’**r**’ , ‘**m**’ 🡪 ’**z**’, ‘’ 🡪 ‘+’, ‘**t**’ 🡪 ‘**g**’, ‘**e**’ 🡪 ’**r**’ , ‘**r**’ 🡪 ’**e**’ , ‘**c**’ 🡪 ’**p**’ , ‘**e**’ 🡪 ’**r**’ , ‘**S**’ 🡪 ’**F**’. After encrypting all letters, the message is: “**Frperg+zrffntr+1$**” and when you reverse it, you get the final encrypted message: “**$1+rtnffrz+greprF**”

### Input

The input data should be read from the console. The input will contain a random number of lines. The line that holds the **keyword “START” or “start”** will always be before the line that holds the **keyword “END” or “end”**. 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 **first line** print the total number of messages that have been sent in format: “**Total number of messages: N**” – where N is the number of encrypted and sent messages.
* On the next N lines print the encrypted messages.
* If **no messages have been sent** between the “**start**” and the “**end**” keywords, you should **print on the console** only one line holding: “**No messages sent.**”

### Constraints

* The **number of messages** between the “**start**” and the “**end**” keywords will be between 0 and 100.
* The **length of each message** will be between 1 and 1000 symbols.
* Each unencrypted message will contain only Latin letters, digits and the special symbols described above.
* Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Comments** | **Output** |
| START  Hello!!!  END | We start conversion from the 1st character: ! 🡪 $, ! 🡪 $, ! 🡪 $, o 🡪 b, l 🡪 y, l 🡪 y, e 🡪 r, H 🡪 U and reverse the newly received string “Uryyb$$$” to the encrypted message “$$$byyrU” | Total number of messages: 1  $$$byyrU |

|  |  |
| --- | --- |
| **Input** | **Output** |
| START  abcdefg  meSSage1  end | Total number of messages: 2  tsrqpon  1rtnFFrz |

|  |  |
| --- | --- |
| **Input** | **Output** |
| start  END | No messages sent. |

|  |
| --- |
| **Input** |
| Normal communication message.  START  Please, try to encrypt the following message!  end |
| **Output** |
| Total number of messages: 1  $rtnffrz+tavjbyybs+rug+gclepar+bg+leg+%rfnryC |

## Tasks

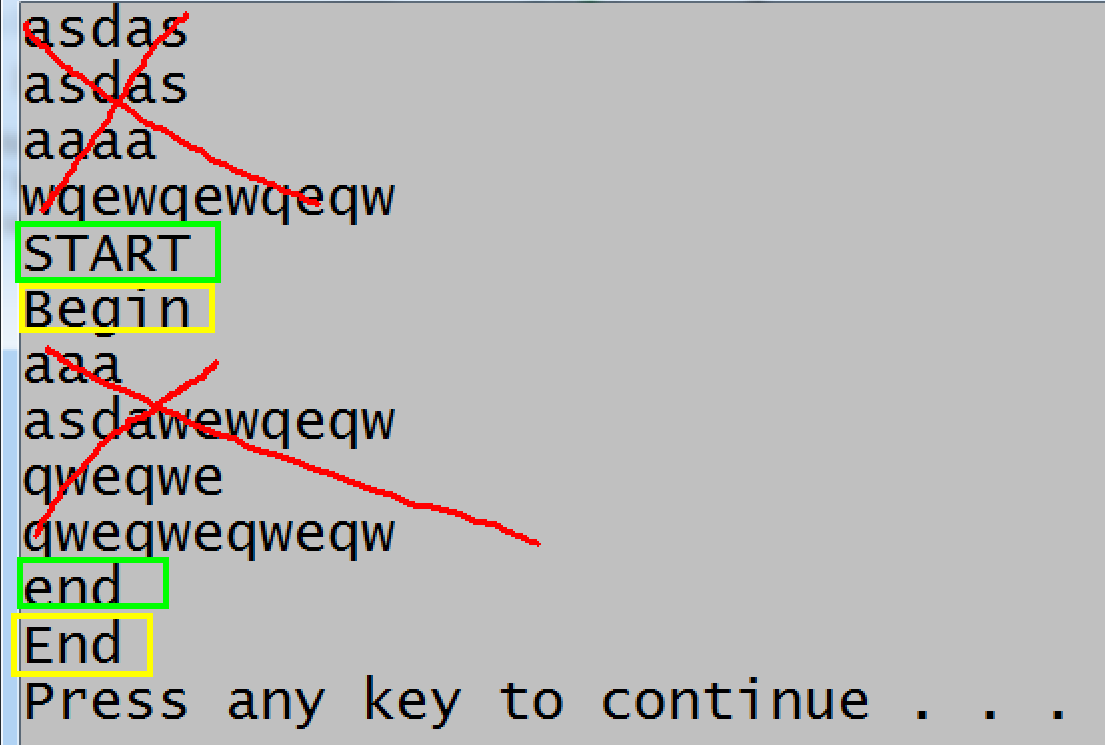
## Step 1: Analyze the problem specifications

Let’s start with analyzing the problem specifications and requirements. Try to figure out how to manage converting the letters from the message into their encrypted versions. For example, you have to convert **A to N, B to O, … and the other way around.** Try to find some kind of pattern for converting each letter into their encrypted version.

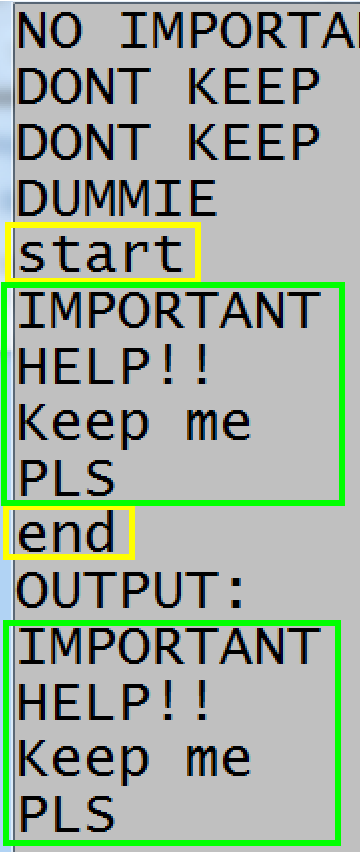
After that, try finding a pattern for converting the special characters as well.   
**Does an actual pattern to do that really exists or do we have to convert them manually?**

## Step 2: Manage reading the input correctly

After the initial analysis of the problem, let’s try handling the first problem that occurs before starting handling all the other major problems – Handling the input. According to the problem description, we have to start parsing the input after **start** and before the **end** command. In that case, let’s try reading some user input from the console and ignore everything that is not in this specified range. For a start, let’s try outputting on the console ‘**Begin**’ after we read the **start** command and output ‘**End**’ after the **end** command. Don’t forget to make your input handling **case-insensitive.** Also, **remember to skip all the empty line messages**. Here is an example:

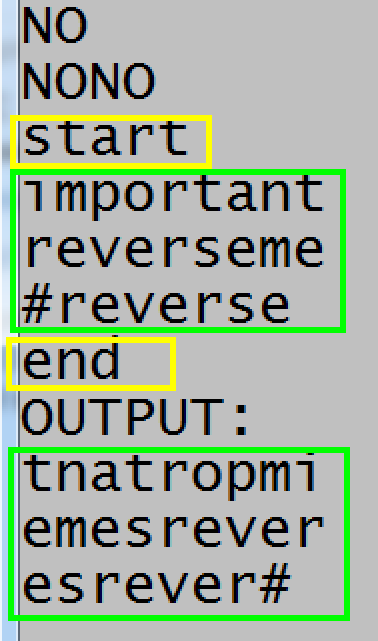


Now, let’s try **keeping all the input** that we receive between then **start** and **end** commands in some way and after we read the **end** command – **output all that we have received on the console**.



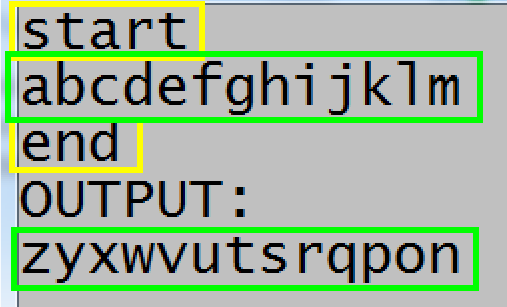
## Step 3: Output the messages received backwards

After we have managed reading the input correctly, let’s begin handling the message manipulation. The first thing we have to do is to handle **reversing the messages we receive** in order to encrypt them correctly. Let’s try managing the previous task again but this time, let’s output the messages **backwards**.

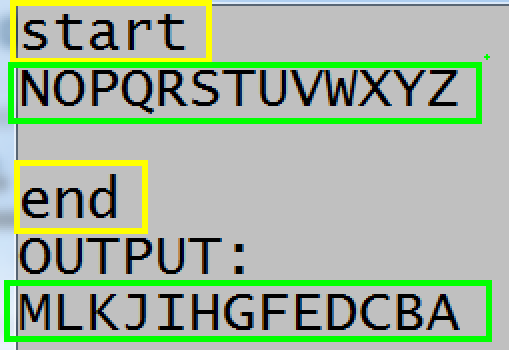


## Step 4: Handle encrypting the letters

Now that we have succeeded in reversing our messages, let’s try encrypting all of their letters. As stated in the problem description, we have to convert the letters from **[A-M] to [N-Z]** and the other way around. **Don’t forget to keep their original casing**. For a start, let’s try encrypting the letters from **[A-M]**. Have you figured out some kind of a pattern to achieve this, or **would you like to use 13 if-else cases**?

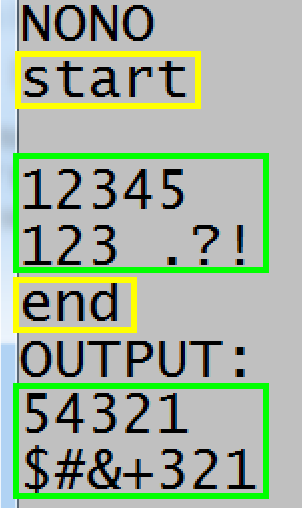


Now let’s try encrypting the letters from **[N-Z]**.



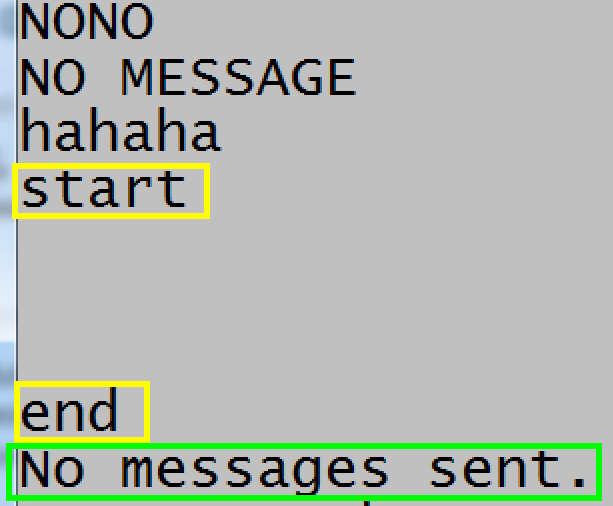
## Step 5: Handle encrypting the digits and special characters

Now that we handled the letters, let’s deal with the special characters and digits as well. As for the digits, they are **not encrypted at all** and stay the same. The special characters, however, need to be handled. As stated in the problem description **the ‘ ’ has to be encrypted as ‘+’ and so on**. Have you figured out some kind of pattern for them or would you like to form **several cases** for them and handle them manually?

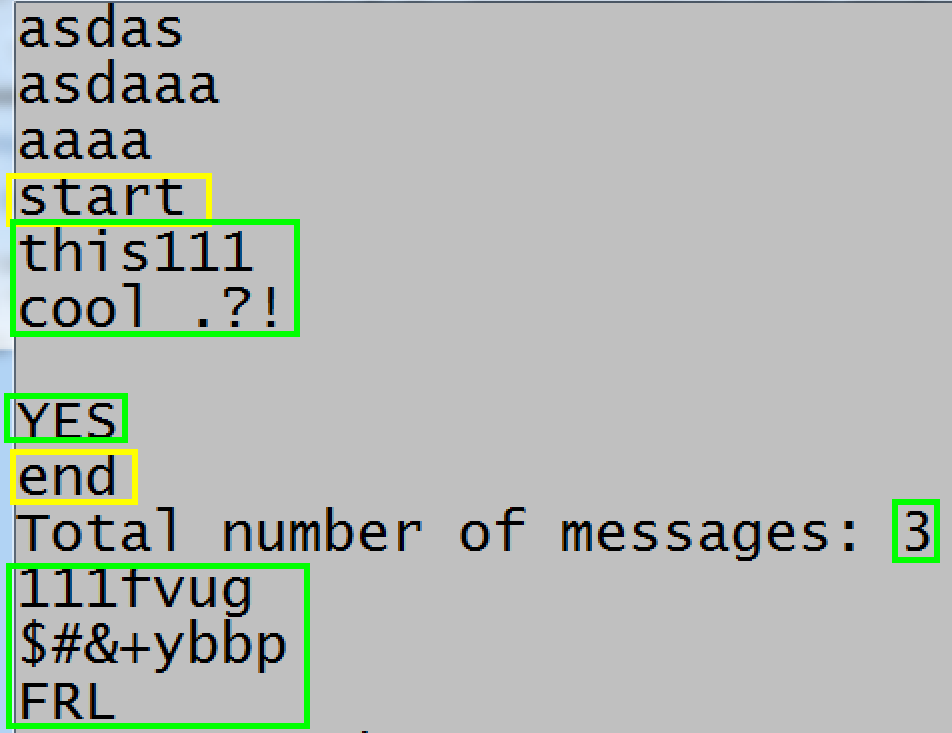


## Step 6: Correctly format the output

We are almost there! Now we have to manage formatting the output. For a start, let’s handle displaying the correct output **when no messages were send whatsoever**.



Now, let’s handle displaying the correct output when there are some messages. First, let’s try to **count all the messages received** in some way in order to display their count in the output. After that, think about **displaying the correct messages** according to the input.



## Step 7: Test your solution and submit it to the Judge System

After completing all the previous steps, we are ready to submit our solution to Judge. Before that, try testing your program with different cases. Try inputting some **border cases**. After that submit to Judge and if you don’t get 100 points, go back to some of the previous steps or try to test your solution again until you have achieved your goal.