2149 The Devil of Gravity

Haven't you ever thought that programs written in Java, C++, Pascal, or any other modern computer languages look rather sparse? Although most editors provide sufficient screen space for at least 80 characters or so in a line, the average number of significant characters occurring in a line is just a fraction. Today, people usually prefer readability of programs to efficient use of screen real estate.

Dr. Faust, a radical computer scientist, believes that editors for real programmers shall be more space efficient. He has been doing research on saving space and invented various techniques for many years, but he has reached the point where no more essential improvements will be expected with his own ideas.

After long thought, he has finally decided to take the ultimate but forbidden approach. He does not hesitate to sacrifice anything for his ambition, and asks a devil to give him supernatural intellectual powers in exchange with his soul. With the transcendental knowledge and ability, the devil provides new algorithms and data structures for space efficient implementations of editors.

The editor implemented with those evil techniques is beyond human imaginations and behaves somehow strange. The mighty devil Dr. Faust asks happens to be the devil of gravity. The editor under its control saves space with magical magnetic and gravitational forces.

Your mission is to defeat Dr. Faust by re-implementing this strange editor without any help of the devil. At first glance, the editor looks like an ordinary text editor. It presents texts in two-dimensional layouts and accepts editing commands including those of cursor movements and character insertions and deletions. A text handled by the devil's editor, however, is partitioned into text segments, each of which is a horizontal block of non-blank characters. In the following figure, for instance, four text segments "`abcdef`", "`ghijkl`", "`mnop`", "`qrstuvw`" are present and the first two are placed in the same row.

```
abcdef  ghijkl
       mnop
qrstuvw
```

The editor has the following unique features.

1. A text segment without any supporting segments in the row immediately below it falls by the evil gravitational force.

2. Text segments in the same row and contiguous to each other are concatenated by the evil magnetic force.

For instance, if characters in the segment "`mnop`" in the previous example are deleted, the two segments on top of it fall and we have the following.

```
abcdef
qrstuvw ghijkl
```

After that, if "`x`" is added at the tail (i.e., the right next of the rightmost column) of the segment "`qrstuvw`", the two segments in the bottom row are concatenated.

```
abcdef
qrstuvwxghijkl
```

Now we have two text segments in this figure. By this way, the editor saves screen space but demands the users' extraordinary intellectual power.

In general, after a command execution, the following rules are applied, where S is a text segment, left(S) and right(S) are the leftmost and rightmost columns of S, respectively, and row(S) is the row number of S.

1. If the columns from left(S) to right(S) in the row numbered row(S)-1 (i.e., the row just below S) are empty (i.e., any characters of any text segments do not exist there), S is pulled down to row(S)-1 vertically, preserving its column position. If the same ranges in row(S)-2, row(S)-3, and so on are also empty, S is further pulled down again and again. This process terminates sooner or later since the editor has the ultimate bottom, the row whose number is zero.

2. If two text segments S and T are in the same row and right(S) +1 = left(T), that is, T starts at the right next column of the rightmost character of S, S and T are automatically concatenated to form a single text segment, which starts at left(S) and ends at right(T). Of course, the new segment is still in the original row.

Note that any text segment has at least one character. Note also that the first rule is applied prior to any application of the second rule. This means that no concatenation may occur while falling segments exist. For instance, consider the following case.

```
    dddddddd
    cccccccc
bbbb
aaa
```

If the last character of the text segment "`bbbb`" is deleted, the concatenation rule is not applied until the two segments "`cccccccc`" and "`dddddddd`" stop falling. This means that "`bbb`" and "`cccccccc`" are not concatenated.

The devil's editor has a cursor and it is always in a single text segment, which we call the current segment. The cursor is at some character position of the current segment or otherwise at its tail. Note that the cursor cannot be a support. For instance, in the previous example, even if the cursor is at the last character of "`bbbb`" and it stays at the same position after the deletion, it cannot support "`cccccccc`" and "`dddddddd`" any more by solely itself and thus those two segments shall fall. Finally, the cursor is at the leftmost "`d`"

The editor accepts the following commands, each represented by a single character.

- `F`: Move the cursor forward (i.e., to the right) by one column in the current segment. If the cursor is at the tail of the current segment and thus it cannot move any more within the segment, an error occurs.

- `B`: Move the cursor backward (i.e., to the left) by one column in the current segment. If the cursor is at the leftmost position of the current segment, an error occurs.

- `P`: Move the cursor upward by one row. The column position of the cursor does not change. If the new position would be out of the legal cursor range of any existing text segment, an error occurs.

- `N`: Move the cursor downward by one row. The column position of the cursor does not change. If the cursor is in the bottom row or the new position is out of the legal cursor range of any existing text segment, an error occurs.

- `D`: Delete the character at the cursor position. If the cursor is at the tail of the current segment and so no character is there, an error occurs. If the cursor is at some character, it is deleted and the current segment becomes shorter by one character. Neither the beginning (i.e., leftmost) column of the current segment nor the column position of the cursor changes. However, if the current segment becomes empty, it is removed and the cursor falls by one row. If the new position would be out of the legal cursor range of any existing text segment, an error occurs.

Note that after executing this command, some text segments may lose their supports and be pulled down toward the hell. If the current segment falls, the cursor also falls with it.

- `C`: Create a new segment of length one immediately above the current cursor position. It consists of a copy of the character under the cursor. If the cursor is at the tail, an error occurs. Also if the the position of the new segment is already occupied by another segment, an error occurs. After executing the command, the created segment becomes the new current segment and the column position of the cursor advances to the right by one column.
- Lowercase and numeric characters (‘`a`’ to ‘`z`’ and ‘`0`’ to ‘`9`’): Insert the character at the current cursor position. The current segment becomes longer by one character. The cursor moves forward by one column. The beginning column of the current segment does not change.

Once an error occurs, the entire editing session terminates abnormally.

Input

The first line of the input contains an integer that represents the number of editing sessions. Each of the following lines contains a character sequence, where the first character is the initial character and the rest represents a command sequence processed during a session. Each session starts with a single segment consisting of the initial character in the bottom row. The initial cursor position is at the tail of the segment. The editor processes each command represented by a character one by one in the manner described above.

You may assume that each command line is non-empty and its length is at most one hundred. A command sequence ends with a newline.

Output

For each editing session specified by the input, if it terminates without errors, your program should print the current segment at the completion of the session in a line. If an error occurs during the session, just print ‘`ERROR`’ in capital letters in a line.

Sample Input

```
3
12BC3BC4BNBBDD5
aaaBCNBBBCb
aaaBBCbNBC
```

Sample Output

```
15234
aba
ERROR
```