BACK

Befunge-93

154,654

DESCRIPTION | MY SOLUTIONS | LEADERBOARD | COMMENTS | README

CODEWRITING SCORE: 300/300

While exploring the ruins of a golden lost city, you discovered an ancient manuscript containing series of strange symbols. Thanks to your profound knowledge of dead languages, you realized that the text was written in one of the dialects of Befunge-93. Looks like the prophecy was true: you are the one who can find the answer to the Ultimate Question of Life! Of course you brought your futuristic laptop with you, so now you just need a function that will run the encrypted message and make you the all-knowing human being.

Befunge-93 is a stack-based programming language, the programs for which are arranged in a two-dimensional *torus* grid. The program execution sequence starts at the top left corner and proceeds to the right until the first direction instruction is met (which can appear in the very first cell). The *torus* adjective means that the program never leaves the grid: when it encounters a border, it simply goes to the next command at the opposite side of the grid.

You need to write a function that will be able to execute the given Befunge-93 `program`. Unfortunately your laptop, futuristic that it is, can't handle more than 10^5 instructions and will probably catch on fire if you try to execute more, so the function should exit after 10^5 commands. The good news is, the prophesy said that the answer to the Ultimate Question of Life contains no more than `100` symbols, so the function should return the `program` output once it contains `100` symbols.

The dialect of Befunge-93 in the manuscript consists of the following commands:

- direction instructions:
  - `>` : start moving right
  - `<` : start moving left
  - `v` : start moving down
  - `^` : start moving up
  - `#` : bridge; skip next cell
- conditional instructions:
  - `_` : pop a `value`; move right if `value = 0`, left otherwise
  - `|` : pop a `value`; move down if `value = 0`, up otherwise
- math operators:
  - `+` : addition; pop `a`, pop `b`, then push `a + b`
  - `-` : subtraction; pop `a`, pop `b`, then push `b - a`
  - `*` : multiplication; pop `a`, pop `b`, then push `a * b`
  - `/` : integer division; pop `a`, pop `b`, then push `b / a`
  - `%` : modulo operation; pop `a`, pop `b`, then push `b % a`
- logical operators:
  - `!` : logical NOT; pop a `value`, if the `value = 0`, push `1`, otherwise push `0`
  - `` ` `` : greater than; pop `a` and `b`, then push `1` if `b > a`, otherwise `0`
- stack instructions:
  - `:` : duplicate value on top of the stack
  - `\` : swap the top stack value with the second to the top
  - `$` : pop value from the stack and discard it
- output instructions:
  - `.` : pop value and output it as an integer followed by a space
  - `,` : pop value and output it as ASCII character
- digits `0-9` : push the encountered number on the stack
- `"` : start string mode; push each character's ASCII value all the way up to the next `"`
- (whitespace character): empty instruction; does nothing
- `@` : end program; the program output should be returned then

If the stack is empty and it is necessary to pop a value, no exception is raised; instead, `0` is produced.

Example

For

```
program = [
    "               v",
    "v  ,,,,,"Hello"<",
    ">48*,          v",
```