Brain transplant

v0.1.0 2025-04-10 MIT-0

Transpile Brainfuck code to Typst

MAJA ABRAMSKI-KRONENBERG
🔗 https://me.digitalwords.net

BRAIN TRANSPLANT provides a Brainfuck-to-Typst transpiler. Let's unpack this mouthful gibberish. *Brainfuck* is a minimalist esoteric programming language that consists of only eight simple instructions yet is Turing-complete (which essentially means everything computable by a fully-fledged language on a computer is in principle computable by it and vice versa). A *transpiler* is a program that translates a program given in a certain programming language to an equivalent program in another language. In our case the package takes a program written in Brainfuck and translates it into Typst code, which can then be displayed or run (evaluated); for example:

```
#import "@preview/brain-transplant:0.1.0": *

#raw(brain-transplant("+++++++[->,.<]", input: "Hi Mom!", evaluate: true))
```

```
Hi Mom!
```

All this has no real practical use, but it's very amusing (mindbogglingly so even 🤯) if you're into programming, minimalism, recreational and artistic use of math and technology, formal rules, meta and the like.

Table of Contents

Introduction 2
I.1 The Brainfuck language 2
I.2 Transpiling 2
I.3 Name 3
I.4 Logo 3
I.5 Copyright and licence 3
I.6 Versioning and stability 3
I.7 Participation and contact 4
Usage 5
Example programs 9
III.1 Printing a string 9
III.2 Making calculations 9
III.3 Processing input 10
III.4 Drawing pictures 10
III.5 Seven-segment display 12
III.6 Self-reference and meta-programming 13

Part I
Introduction

I.1 The Brainfuck language

The world of esoteric programming languages (*esolangs*) is a wonderland of unpractical yet eye-opening languages which seem odd at first glance and odder in a deeper look. Brainfuck (created in 1993 by Urban Müller°[1]) is one of the better known of the lot. It uses a simple machine model that consists of a program, an instruction pointer (also known as *program counter* or PC), a one-dimensional array, a movable data pointer and simple input and output capabilities. It has eight instructions, as follows:

arithmetics	`+`	**increment** the **value** of the cell indicated by the pointer
	`-`	**decrement** the **value** of the cell indicated by the pointer
memory access	`>`	**move** the pointer to the **next** cell
	`<`	**move** the pointer to the **previous** cell
control flow	`[`	**open** a **loop** that repeats until the cell indicated by the pointer is `0`; a more verbose description: if the data at the pointer is zero, then instead of moving the program counter forward to the next instruction, jump it forward to the instruction after the matching `]` instruction
	`]`	**close** a **loop**; a more verbose description: if the data at the pointer is nonzero, then instead of moving the instruction pointer forward to the next instruction, jump it back to the instruction after the matching `[` instruction
I/O	`.`	**output** the data at the pointer (normally prints its ASCII value)
	`,`	read one item from the **input** and store it in the cell indicated by the pointer

The interesting thing is that this minimal system can be proven to be the identical to a fully-fledged programming language in terms of what is computable. In computer science jargon, it's said to be *Turing-complete*. This doesn't means it's *easy*, *practical* or *efficient* to use Brainfuck for anything (i.e. anything beyond the joy of creating and appreciating neat little programs), but it's *possible* to use it for all computational tasks solvable by a computer.

If this resonates with you I wholeheartedly suggest you read the book *Gödel, Escher, Bach: an Eternal Golden Braid* by Douglas Hofstadter. Also consider embracing the rabbit hole and delving into esolangs; this talk° is a nice introduction.

I.2 Transpiling

There are several ways to make code written in any programming language run on a machine:

[1]External links in this document are denoted with a red circle, following Butterick's *Practical typography*°.

- One common way is to *compile* it, which means a program (a *compiler*) converts the high-level code to machine code ('1s and 0s' as it's commonly put in non-technical terms) that can run on the hardware directly. Languages such as C and Rust are canonically compiled.
- Another common strategy is to *interpret* it on the fly, which means a program (an *interpreter* this time, as you've guessed astutely…) reads the code and acts as it instructs, mediating between the code and the computer while it runs the program. Languages such as Python and Lua are canonically interpreted.
- Yet another strategy is to *transpile* (translate) the code to another language, which then can be compiled or interpreted by proper compilers and interpreters. The process is also known as *source-to-source compiling* or *transcompiling*. TypeScript is canonically transpiled, to JavaScript.

There's already a Brainfuck interpreter written in Typst, called ESOTEFY°, but I wanted to implement a *transpiler*, ergo BRAIN TRANSPLANT, which provides a function that takes a Brainfuck program as its input and produces a Typst program that preforms the same task. I got the inspiration while reading this gemlog post° (also available on the web°; code°) when I couldn't sleep and was so excited by how straightforward it is to transpile Brainfuck to C that I decided to make one for Typst, written in Typst, first thing in the morning. It was indeed very straightforward to implement; writing this documentation took *way* more time to write than the package itself…

I.3 Name

The name BRAIN TRANSPLANT combines the name of the programming language (Brainfuck) with the process the package performs (transpiling). The medical procedure of transplanting a brain — or, from the other perspective, a whole-body transplant — is not practised in humans, but the concept is explored in science fiction.

I.4 Logo

The logo features a minimalist icon of a brain with < on the left hemisphere and > on the right; see § I.5 for attribution. The background colour is the same shade of turquoise used by MANTYS.

I.5 Copyright and licence

BRAIN TRANSPLANT is released under MIT-0°.

The logo features an image° by Arief Mochjiyat° which is released under CC BY-3.0. I attribute them willingly, as I find the graphics very fitting for the logo. Go check their other icons°.

I.6 Versioning and stability

BRAIN TRANSPLANT follows the Semantic Versioning scheme (SemVer 2.0.0°). While it is fully usable in its current form (version `0.*.*`), changes to the API might occur in future versions. This should not pose a problem:

- When you import a package in Typst you can indicate the version (for example, `#import "@preview/example:0.1.0"`), so no surprises should occur.
- Changes to the API will be clearly indicated.

I.7 Participation and contact

If there is anything that doesn't work well or any feature you want added or changed, don't hesitate to open an issue° on the Git repository, and I will do my best to make the package more useful for you and others. If you want to contribute code or documentation (changes, additions, corrections, improvements, etc. no matter how small or large), patches° ('pull requests') are very welcome; thanks! The repository uses Radicle, a P2P distributed Git forge; for more information, see their website° and the user guide° (in particular, the sections about issues° and patches°).

Not everyone is familiar with Git and Radicle, so if that poses a problem feel free to contact me in any other way; see `https://me.digitalwords.net/`° for contact information.

Part II

Usage

In order to 0.1.0

```
#brain-transplant(
  ⟨code⟩,
  ⟨input⟩: "",
  ⟨memsize⟩: 30000,
  ⟨evaluate⟩: true,
  ⟨unsafe⟩: false,
  ⟨numbf⟩: false
) → str
```

Transpile a Brainfuck program to Typst.

Argument

`⟨code⟩` `str`

The Brainfuck code. All characters save `+-<>[].,#`, including line breaks, are ignored and thus can be used as comments. `#` (a non-standard instruction) prints debug information: the position of the pointer, the value it indicates, the position in the input and the values of the ten (at most) first cells.

```
#raw(brain-transplant(",>,>,#", input: "Hi!"))
```

```
p      = 2
mem[p] = 33
in-pos = 3
mem[0] = 72
mem[1] = 105
mem[2] = 33
mem[3] = 0
mem[4] = 0
mem[5] = 0
mem[6] = 0
mem[7] = 0
mem[8] = 0
mem[9] = 0
```

Argument

`⟨input⟩: ""` `str`

The input read by the program. In order to input specific numbers, use `\u{…}`; for example, for programs which reads `0`-terminated input, end the input with `\u{0}`. Note that the numbers are written in hexadecimal basis. For more information read Typst's documentation°.

Argument

⟨memsize⟩: 30000 int

The size of the one-dimensional array for the use of the program.

Argument

⟨evaluate⟩: true bool

Whether to evaluate (run) the transpiled program and return its output (default, `true`) or output the transpiled program without evaluating it (the output can be saved into a variable and evaluated later using the `eval()` function.).

Consider this simple program that outputs ‘`Hi Mom!`’. When ⟨evaluate⟩ is set to `true` (or left unset), it behaves like this:

```
#raw(brain-transplant("+++++++[->,.<]", input: "Hi Mom!", evaluate: true))
```

```
Hi Mom!
```

However, when it is set to `false`, it behaves like this:

```
#raw(brain-transplant("+++++++[->,.<]", input: "Hi Mom!", evaluate: false))
```

```
let mem = array(range(0, 30000)).map(a => 0)
let p = 0
let input = "Hi Mom!"
let input-pos = 0
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
mem.at(p) = calc.rem-euclid(mem.at(p) + 1, 256)
while (mem.at(p) != 0) {
mem.at(p) = calc.rem-euclid(mem.at(p) - 1, 256)
p += 1
assert(p < 30000, message: "🧟 BRAIN TRANSPLANT REJECTED: the data pointer of
brain-transplant exceeded its maximum value (29999)")
assert(input-pos < 7, message: "🧟 BRAIN TRANSPLANT REJECTED: tried to read
more input than what was provided")
mem.at(p) = calc.rem-euclid(str.to-unicode(input.at(input-pos)), 256)
input-pos += 1
str.from-unicode(calc.rem-euclid(mem.at(p), 256))
p -= 1
assert(p >= 0, message: "🧟 BRAIN TRANSPLANT REJECTED: the data pointer of
brain-transplant its exceeded minimum value (0)")
}
```

The first part of the code is a preamble that binds° required variables, and after it the commands are transpiled one after the other, with added `assert()` functions which are triggered by ⟨unsafe⟩ being set to `false` by default, following Rust's behaviour.

Argument

⟨unsafe⟩: `false` `bool`

If set to `"false"`, safeguards limiting the pointer from exceeding its minimum and maximum values are provided. Otherwise, the behaviour is unpredictable in such cases. Compare this:

```
#raw(brain-transplant("><", evaluate: false, unsafe: true))
```

```
let mem = array(range(0, 30000)).map(a => 0)
let p = 0
let input = ""
let input-pos = 0
p += 1
p -= 1
```

with this:

```
#raw(brain-transplant("><", evaluate: false, unsafe: false))
```

```
let mem = array(range(0, 30000)).map(a => 0)
let p = 0
let input = ""
let input-pos = 0
p += 1
assert(p < 30000, message: "🧟 BRAIN TRANSPLANT REJECTED: the data pointer of brain-transplant exceeded its maximum value (29999)")
p -= 1
assert(p >= 0, message: "🧟 BRAIN TRANSPLANT REJECTED: the data pointer of brain-transplant its exceeded minimum value (0)")
```

Argument

⟨numbf⟩: false bool

Named after NumPy°, this option breaks the limit of one byte (0 to $2^8 - 1$) per cell and unlocks larger numbers as well as negative numbers. It also changes the output to show as numbers, not ASCII characters.

```
#raw(brain-transplant("-.", numbf: true))
```

-1

```
#raw(brain-transplant(",+.", numbf: true, input: "\u{51F}"))
```

1312

Part III
Example programs

III.1 Printing a string

Let’s begin with a classic ‘Hello, World!’ program:

```
#raw(brain-transplant(">++++++++[<+++++++++>-]<.>++++[<+++++++>-]<+.+++++++..+++.>>++++++[<+++++++>-]<+
+.------------.>++++++[<+++++++++>-]<+.<.+++.------.--------.>>>++++[<++++++++>-]<+."))
```

```
Hello, World!
```

The above program encodes the string within the code itself; the following uses the input functionality (each `,`-instruction reads one letter):

```
#raw(brain-transplant("+++++++[->,.<]", input: "Hi Mom!"))
```

```
Hi Mom!
```

III.2 Making calculations

This program outputs square numbers from 0 to 10000 (source°):

```
#table(
  stroke: none,
  columns: (1fr,) * 16,
  ..brain-transplant("++++[>+++++<-]>[<+++++>-]+<+[>[>+>+<<-]++>>[<<+>>-]>>>[-]++>[-]+>>>+[[-]++++++>>>]<<<[[<++++++
++<++>>-]+<.<[>----<-]<]<<[>>>>>[>>>[-]+++++++++<[>-<-]+++++++++>[-[<-
>-]+[<<<]]<[>+<-]>]<<-]<<-]").split("\n").map(raw)
)
```

```
0     1     4     9     16    25    36    49    64    81    100   121   144   169   196   225
256   289   324   361   400   441   484   529   576   625   676   729   784   841   900   961
1024  1089  1156  1225  1296  1369  1444  1521  1600  1681  1764  1849  1936  2025  2116  2209
2304  2401  2500  2601  2704  2809  2916  3025  3136  3249  3364  3481  3600  3721  3844  3969
4096  4225  4356  4489  4624  4761  4900  5041  5184  5329  5476  5625  5776  5929  6084  6241
6400  6561  6724  6889  7056  7225  7396  7569  7744  7921  8100  8281  8464  8649  8836  9025
9216  9409  9604  9801  10000
```

This one outputs the first numbers in the Fibonacci sequence° (source°):

```
#raw(brain-transplant(">+>+<<+++++++++[>++++++<-]>.<++++++++[>------<-]++++++++[>>++++<<<-]>>>.<<<+++++++++[>>>----
<<<-]++++++++[>>++++++<<-]>>.<<+++++++++[>>------<<-]++++[>>[>+>+<<-]<[>>>+<<<-]>>[<<+>>-]>[<<+>>-]<++++++++[>+++
+<-]>.<++++++++[>----<-]++++++++[<++++++>-]<.>++++++++[<------>-]<<<-]"))
```

```
1 1 2 3 5 8
```

III.3 Processing input

This program counts linebreaks, words and characters (source°):[2]

```
#raw(brain-transplant(">>>+>>>>>+>>+>>+[<<],[-[-[-[-[-[-[-[-[<+>-[>+<-[>-<-[-[-[<++[<++++++>-]<[>>[-<]<[>]<-]>>[<+>-
[<->[-]]]]]]]]]]]]]]]]<[-<<[-]+>]<<[>>>>>+<<<<<<-]>[>]>>>>>>>+>[<+[>+++++++++<-[>-<-]++>[<+++++++>-[<-
>-]+[+>>>>>>]]<[>+<-]>[>>>>>++>[-]]+<]>[-<<<<<<]>>>>],]+<++>>>[[+++++>>>>>>]<+>+[[<++++++++>-]<.<<<<<]>>>>>>>>]",
input: "My dreams hold diamond nightmares
Salvation holds the sea
The night holds silent echoes
Your love is meaningless to me

Rock my dreams like a lovestruck smile
Rock my dreams with your love, your love
Rock the night like a heart-break missile
Rock the night with your love, your love.

My dreams are with the night
Whisper my dreams with salvation\u{0}"))
```

```
  11   61   346
```

This one sorts the input (source°):

```
#raw(brain-transplant(">>,[>>,]<<[[-<+<]>[>[>>]<[.[-]<[[>>+<<-]<]>>]>]<<]", input: "abracadabra\u{0}"))
```

```
aaaaabbcdrr
```

III.4 Drawing pictures

Printing to the screen, doing math and processing input is all fine and dandy, but have you heard about Brainfuck's stunning graphic capabilities?

This program draws the Sierpiński triangle° (source°):[3]

[2]The lyrics, by the way, are a computer program written in the Rockstar° programming language. Maybe one day I will write a transpiler or an interpreter for Rockstar in Typst. h/t Q0° ☺

[3]See also this implementation°, where the program is nicely typeset in the form of the Sierpiński triangle (spaces are ignored).

```
#raw(brain-transplant("++++++++[>+>++++<<-]>++>>+<[-[>>+<<-]+>>]>+[-<<<[->[+[-]+>++>>>-<<]<[<]>>++++++[<<++++
+>>-]+<<++.[-]<<]>.>+[>>]>+]"))
```

```
                               *
                              * *
                             *   *
                            * * * *
                           *       *
                          * *     * *
                         *   *   *   *
                        * * * * * * * *
                       *               *
                      * *             * *
                     *   *           *   *
                    * * * *         * * * *
                   *       *       *       *
                  * *     * *     * *     * *
                 *   *   *   *   *   *   *   *
                * * * * * * * * * * * * * * * *
               *                               *
              * *                             * *
             *   *                           *   *
            * * * *                         * * * *
           *       *                       *       *
          * *     * *                     * *     * *
         *   *   *   *                   *   *   *   *
        * * * * * * * *                 * * * * * * * *
       *               *               *               *
      * *             * *             * *             * *
     *   *           *   *           *   *           *   *
    * * * *         * * * *         * * * *         * * * *
   *       *       *       *       *       *       *       *
  * *     * *     * *     * *     * *     * *     * *     * *
 *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Interestingly, this diamond shape, which to us seems simpler, takes more instructions to draw than the above fractal (source°):

```
#raw(brain-transplant("+++++++[>+>++++++>+++++>+<<<<-]>+
++>>--->++++[->>>++++++++[<+++>-]<<<[<.>>+>--<<-]>>[<<<<.
>>>>-]<<<<<.>>>>[<+>-]<]++++++++++[->++++++++++>+<<[>->+
+<<-]++++++++++>[<<.>->-]>[<<<<.>>>>-]<<<<<.>>>]"))
```

```
          *
         ***
        *****
       *******
      *********
     ***********
    *************
   ***************
  *****************
 *******************
*********************
 *******************
  *****************
   ***************
    *************
     ***********
      *********
       *******
        *****
         ***
          *
```

III.5 Seven-segment display

So we had printing text (§ III.1) and drawing graphics (§ III.4). Why not combine them both? This program, the longest and most complex so far, can print out strings that consist of `()-./0123456789abcdef␣` as slanted seven-segment display (source°):

```
#let numwarp = "
>>>>+>+++>+++>>>>>+++[>,+>++++[>++++<-]>[<<[-
[->]] >[<]>-]<<[>+>+>>+>+[<<<<]<+>>[+<]<[>]>+
[[>>>]>>+[< <<<]>-]+<+>>>-[<<+[>]>>+<<<+<+<--
------[<<-<<+[>]>+<<-<<-[<<<+<-[>>]<-<-<<<-<-
---[<<<->>>>+<-[<<<+[>]>+<<+<-<-[<<+<-<+[>>]<
+<<<<+<-[<<-[>]>>-<<<-<-<-[<<<+<-[>>]<+<<<+<+
<-[<<<<+[>]<-<<-[<<+[>]>>-<<<<-<-[>>>>>+<-<<<
+<-[>>+<<-[<<-<-[>]>+<<-<-<-[<<+<+[>]<+<+<-[>
>-<-<-[<<-[>]<+<++++[<-------->-]++<[<<+[>]>>
-<-<<<<-[<<-<<->>>>-[<<<<+[>]>+<<<<-[<<+<<-[>
>]<+<<<<<-[>>>>-<<<-<-]]]]]]]]]]]]]]]]]]]]]]>
[>[[[<<<<]>+>>[>>>>>]<-]<]>>>+>>>>>>>+>]<]<[-
]<<<<<<<++<+++<+++[[>]>>>>>>++++++++[<<++++>+
+++++>-]<-<<[-[<+>>.<-]]<<<<[-[-[>+<-]>]>>>>>
[.[>]]<<[<+>-]>>>[<<++[<+>--]>>-]<<[->+<[<++>
-]]<<<[<+>-]<<<<]>>+>>>--[<+>---]<.>>[[-]<<]<
]"
#raw(brain-transplant(numwarp, input: "ea7 f00d\u{0}"))
#raw(brain-transplant(numwarp, input: "7ea a17d
(0ffee\u{0}"))
```

```
               \
               /\
            /\ \/
            \ \
          /\ \/
          \ \
        /  \/
        \/
         \

    /\
      \
  /\
   /\
/\ \/
\/
 \/
                            /\
                            \/
                          /\ \/
                          \/
                        /  \/
                        \/
                      /  \
                      \/
                    /\ \
                    \ \
                  /  \/
                  \
                   \/

               \
               /\
            /\ \/
              \
           \
            \
        /\
         /\
         \/

    /\
     /\
  /\ \/
  \/
/\ \/
  \
```

It’s beyond me how this program works. Pure dark magic!

Fun fact: tea leaves and coffee beans are edible, so one can indeed eat tea and coffee…

III.6 Self-reference and meta-programming

I saved for the end the most spectacular, breathtaking and marvellous Brainfuck programs I know of.

Let's begin with a program whose output is itself, also known as a quine°. The most trivial and underwhelming program that satisfies this requirement is the empty program, which outputs an empty string.

```
#raw(brain-transplant(""))
```

```
```

The shortest non-trivial quine is 391 instructions long, at least as far as I know (source°):

```
#raw(brain-transplant("->++>+++>+>+>+++>>>>>>>>>>>>>>>>>>>>+>+>++>+++>++>>++
+>+>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+>+>>+++>>+++>>>>>+++>+>>>>>>>>>++>+++>+++>+>>+++>>>+++>+>++>+++>>>+>+>++>++
+>+>+>>+++>>>>>>>+>+>>>+>+>++>+++>+++>+>>+++>>>+++>+>++>+++>++>>+>+>++>+++>+>+>>+++>>>>>+++>+>>>>>++>+++>+++>+>>++
+>>>+++>+>+++>+>>+++>>+++>>++[[>>+[>]++>++[<]<-]>+[>]<+<+++[<]<+]>+[>]++++>++[[<++++++++++++++++>-]<+++++++++.<]"))
```

```
->++>+++>+>+>+++>>>>>>>>>>>>>>>>>>>>+>+>++>+++>++>>+++>+>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+>+>>+++>>+++>>>>>++
+>+>>>>>>>>>++>+++>+++>+>>+++>>>+++>+>++>+++>>>+>+>++>+++>+>+>>+++>>>>>>>+>+>>>+>+>++>+++>+++>+>>+++>>>+++>+>++>++
+>++>>+>+>++>+++>+>+>>+++>>>>>+++>+>>>>>++>+++>+++>+>>+++>>>+++>+>+++>+>>+++>>+++>>++[[>>+[>]++>++[<]<-]>+[>]<+<++
+[<]<+]>+[>]++++>++[[<++++++++++++++++>-]<+++++++++.<]
```

As expected, if we feed the transpiler the output produced by this program, the output will be the program itself yet again:

```
#raw(brain-transplant(brain-transplant("->++>+++>+>+>+++>>>>>>>>>>>>>>>>>>>>+>+>++>+++>++>>++
+>+>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+>+>>+++>>+++>>>>>+++>+>>>>>>>>>++>+++>+++>+>>+++>>>+++>+>++>+++>>>+>+>++>++
+>+>+>>+++>>>>>>>+>+>>>+>+>++>+++>+++>+>>+++>>>+++>+>++>+++>++>>+>+>++>+++>+>+>>+++>>>>>+++>+>>>>>++>+++>+++>+>>++
+>>>+++>+>+++>+>>+++>>+++>>++[[>>+[>]++>++[<]<-]>+[>]<+<+++[<]<+]>+[>]++++>++[[<++++++++++++++++>-]<+++++++++.<]")))
```

```
->++>+++>+>+>+++>>>>>>>>>>>>>>>>>>>>+>+>++>+++>++>>+++>+>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+>+>>+++>>+++>>>>>++
+>+>>>>>>>>>++>+++>+++>+>>+++>>>+++>+>++>+++>>>+>+>++>+++>+>+>>+++>>>>>>>+>+>>>+>+>++>+++>+++>+>>+++>>>+++>+>++>++
+>++>>+>+>++>+++>+>+>>+++>>>>>+++>+>>>>>++>+++>+++>+>>+++>>>+++>+>+++>+>>+++>>+++>>++[[>>+[>]++>++[<]<-]>+[>]<+<++
+[<]<+]>+[>]++++>++[[<++++++++++++++++>-]<+++++++++.<]
```

The following program also outputs a Brainfuck program, but this time what it outputs is a program that prints a given `0`-terminated string (cf. § III.1; source°), for example:

```
#text(size: 0.9em, raw(brain-transplant("+++++[>+++++++++<-],[[>--.++>+<<-]>+.->[<.>-]<<,]", input: "Eat your
brains!\u{0}")))
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.---------------------------------------------------------------------++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++
+.-------------------------------------------------------------------------------------------------++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.--------------------------------------------------------------------------------------------------------------------+++++++++++
++++++++++++++++++++.--------------------------------+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++
+.-------------------------------------------------------------------------------------------------------------------------++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.---------------------------------------------------------------------------------------------------------------++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.---------------------------------------------------------------------------------------------------------------------++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.------------------------------------------------------------------------------------------------------------------+++++++++++++
++++++++++++++++++.--------------------------------+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++.--------------------------------------------------------------------------------------------------+++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.------------------------------------------------------------------------------------------------------------------+++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.-------------------------------------------------------------------------------------------------++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.---------------------------------------------------------------------------------------------------------++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.--------------------------------------------------------------------------------------------------------------+++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+.-------------------------------------------------------------------------------------------------------------------++++++++++++
++++++++++++++++++++.---------------------------------
```

Now, if we feed the transpiler the output, it will output the original string!

```
#raw(brain-transplant(brain-transplant("+++++[>+++++++++<-],[[>--.++>+<<-]>+.->[<.>-]<<,]", input: "Eat your brains!
\u{0}")))
```

```
Eat your brains!
```

⚠ Notice! While you can eat tea leaves and coffee beans (§ III.5), you are strongly discouraged from eating your own, or anyone else’s for that matter, brains 🧠