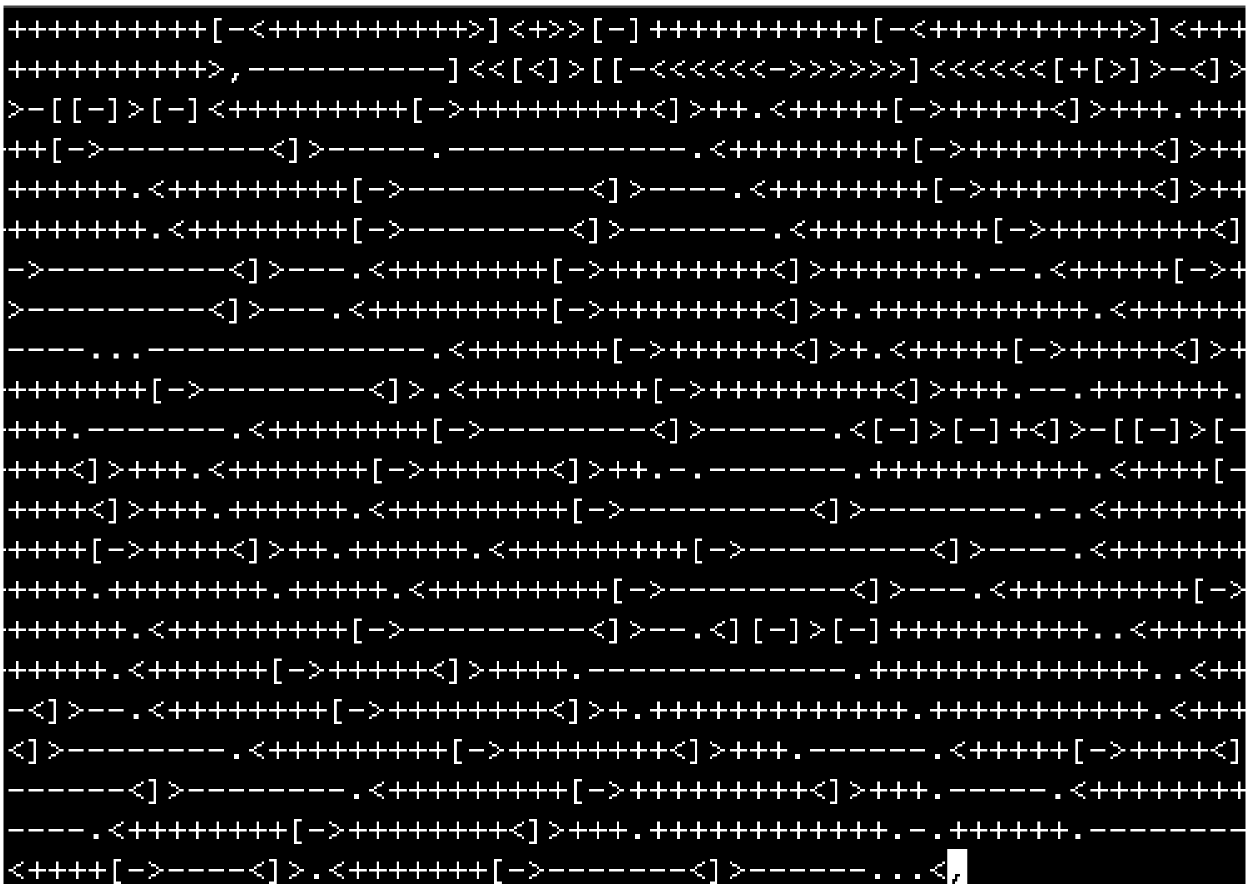
Brainf*ck

Meet the most frustrating programming language ever


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

Report by

Mr. Worakun Ata, Code ID: 600510528

Mr. Jakkrit Boonnet, Code ID: 600510533

Mr. Jirajate Jantarawong, Code ID: 600510534

In the course

Organization of Programming Languages

Preface

Hello, this report is a piece project teamwork of Organization of Programming Languages course. The purpose of report is for study the any interesting programming languages in computer science such as syntax, structure, how to work, advance grammar, or presenting the interesting of some programming languages.

For this report, we present the brainfuck or brainfsck. it's an esoteric programming language. the interesting thing of this language is it has super small compiler, base on Turing complete, it isn't intended for practical use, but to challenge and amuse programmers.

If you are or want to be or going to be or similarly anything like programmer. Hope you enjoy too yeah!

Content

Brainfuck is the most famous esoteric programming language, and has inspired the creation of a host of other languages. Due to the fact that the last half of its name is often considered one of the most offensive words in the English language, it is sometimes referred to as brainf***, brainf*ck, brainfsck, b****fuck (as a joke), brainf**k, branflakes, or BF. This can make it a bit difficult to search for information regarding brainfuck on the web, as the proper name might not be used at all in some articles.

1 – Language overview

Brainfuck operates on an array of memory cells, also referred to as the tape, each initially set to **zero**. There is a pointer, initially to the first memory cell. The commands are:

Command	Description
>	Move the pointer to the right
<	Move the pointer to the left
+	Increment the memory cell under the pointer
–	Decrement the memory cell under the pointer
.	Output the character signified by the cell at the pointer
,	Input a character and store it in the cell at the pointer
[	Jump past the matching] if the cell under the pointer is 0
]	Jump back to the matching [if the cell under the pointer is nonzero

All characters other than `><+-.,[]` should be considered comments and ignored. But, see extensions below.

1.1 – Language design

The language consists of eight commands, listed before page. A brainfuck program is a sequence of these commands, possibly interspersed with other characters (which are ignored). The commands are executed sequentially, with some exceptions: an instruction pointer begins at the first command, and each command it points to is executed, after which it normally moves forward to the next command. The program terminates when the instruction pointer moves past the last command.

The brainfuck language uses a simple machine model consisting of the program and instruction pointer, as well as an array of at least 30,000 byte cells initialized to zero; a movable data pointer (initialized to point to the leftmost byte of the array); and two streams of bytes for input and output (most often connected to a keyboard and a monitor respectively, and using the ASCII character encoding).

Brainfuck programs can be translated into C using the following substitutions, assuming **ptr** is of type **char*** and has been initialized to point to an array of zeroed bytes:

brainfuck command	C equivalent

(Program Start)	char array[INFINITY_LARGE_SIZE] = {0};
	char *prt = array;

>	++ptr;
<	--ptr;
+	++(*ptr);
-	--(*ptr);
.	putchar(*ptr);
,	*ptr=getchar();
[	while(*ptr){
]	}

2 – History

Brainfuck was invented by **Urban Müller** in 1993, in an attempt to make a language for which he could write the smallest possible compiler for the Amiga OS, version 2.0. He managed to write a 240-byte compiler. The language was inspired by **FALSE**, which had a 1024-byte compiler. Müller chose to name the language brainfuck (with the initial letter in lower case, although it is now often capitalised).

It is not known to what extent **Müller** was aware of or influenced by Böhm's language P'' published in 1964, of which brainfuck can be considered a minor variation.

3 – Examples

Hello, World! This program print out the words *Hello World!* :

```
+++++ +++++             initialize counter (cell #0) to 10
[                       use loop to set 70/100/30/10
   > +++++ ++              add  7 to cell #1
   > +++++ +++++           add 10 to cell #2
   > +++                   add  3 to cell #3
   > +                     add  1 to cell #4
<<<< -                  decrement counter (cell #0)
]
> ++ .                  print 'H'
> + .                   print 'e'
+++++ ++ .              print 'l'
.                       print 'l'
+++ .                   print 'o'
> ++ .                  print ' '
<< +++++ +++++ +++++ .  print 'W'
> .                     print 'o'
+++ .                   print 'r'
----- - .               print 'l'
----- --- .             print 'd'
> + .                   print '!'
> .                     print '\n'
```

The same program in minimised form:

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

to see the result : https://fatiherikli.github.io/brainfuck-visualizer/

Currently, the shortest known program printing Hello, World! is written by KSab from https://codegolf.stackexchange.com/a/163590/59487.

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

4 – Algorithms

In the interest of generality, the algorithms will use variable names in place of the < and > instructions. Temporary cells are denoted "temp". When using an algorithm in a program, replace the variable names with the correct number of < or > instructions to position the pointer at the desired memory cell.

Example:

If "a" is designated to be cell **1**, "b" is cell **4**, and the pointer is currently at cell 0, then:

```
a[b+a-]
```

becomes:

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

If a particular algorithm requires cell value wrapping, this will be noted, along with a non-wrapping version, if known. Certain assumptions, such as that a temporary memory cell is already zero, or that a variable used for computation can be left zeroed, are not made. Some optimizations can therefore be performed when exact conditions are known.

4.1 – Header comment

The usefulness of this type of comment is that instructions commonly used for punctuation (such as "," and ".") may be used freely. The use of "[" and "]" inside a comment should be avoided, unless they are matched. This commenting style does not work well for internal code comments, unless strategically placed where the cell value is known to be zero (or can be modified to be zero and restored):

```
[comment]
```

To make no assumption about the initial cell value, use:

```
[-][comment]
```

Since loops only terminate when/if the current cell is zeroed, comments can safely be placed directly behind any other loop.

```
,[.,][comment]
```

4.2 – Read all characters into memory

```
,[>,]
```

4.3 – Read until newline/other char

```
----------[++++++++++>,----------]++++++++++
```

Adjust the number of +/– to the char code you want to match. Omit the final + block to drop the char. Requires wrapping.

4.4 – Read until any of multiple chars

```
+[>,
 (-n1)[(+n1)[>+<-]]>[<+>-]<
 (-n2)[(+n2)[>+<-]]>[<+>-]<
 etc
]
```

(+/–n1) means repeat that operator n1 times. n1, n2 etc are the char codes you want to match. One line for each.

4.5 : x = 0

```
x[-]
```

4.6 : x = y

```
temp0[-]
x[-]
y[x+temp0+y-]
temp0[y+temp0-]
```

4.7 : x = x + y

```
temp0[-]
y[x+temp0+y-]
temp0[y+temp0-]
```

4.8 : x = x – y

```
temp0[-]
y[x-temp0+y-]
temp0[y+temp0-]
```

4.9 : x = x * y

```
temp0[-]temp1[-]
x[temp1+x-]
temp1[
y[x+temp0+y-]temp0[y+temp0-]
temp1-]
```

4.10 : x = x * x

Same as above, but copies x to temp2. (Might be improvable)

```
temp0[-]temp1[-]temp2[-]
x[temp2+temp1+x-]
temp1[
 temp2[x+temp0+temp2-]
 temp0[temp2+temp0-]
 temp1-
]
```

4.11 : x = x / y

Attribution: Jeffry Johnston

```
temp0[-]temp1[-]temp2[-]temp3[-]
x[temp0+x-]
temp0[
y[temp1+temp2+y-]
temp2[y+temp2-]
temp1[
 temp2+
 temp0-[temp2[-]temp3+temp0-]
 temp3[temp0+temp3-]
 temp2[
  temp1-
  [x-temp1[-]]+
 temp2-]
temp1-]
x+
temp0]
```

4.12 : x = x ^ y

Attribution: chad3814

```
temp0[-]
x[temp0+x-]
x+
y[
  temp1[-]temp2[-]
  x[temp2+x-]
  temp2[
    temp0[x+temp1+temp0-]
    temp1[temp0+temp1-]
  temp2-]
y-]
```

4.13 : Find a zeroed cell

To the right

```
[>]
```

To the left

```
[<]
```

4.14 : x = x == y

Attribution: Jeffry Johnston

The algorithm returns either 0 (false) or 1 (true) and preserves y.

```
temp0[-]temp1[-]
x[temp1+x-]+
y[temp1-temp0+y-]
temp0[y+temp0-]
temp1[x-temp1[-]]
```

Attribution: Yuval Meshorer

Sets x to be 1 if x == y, 0 otherwise.

```
x[
y-x-]
y[[-]
x+y]
```

And if you don't need to preserve x or y, the following does the task without requiring any temporary blocks. Returns 0 (false) or 1 (true).

```
x[-y-x]+y[x-y[-]]
```

4.15 : x = x != y

Attribution: Jeffry Johnston

The algorithm returns either 0 (false) or 1 (true).

```
temp0[-]
temp1[-]
x[temp1+x-]
y[temp1-temp0+y-]
temp0[y+temp0-]
temp1[x+temp1[-]]
```

4.16 : x = x < y

Attribution: Ian Kelly

x and y are unsigned. temp1 is the first of three consecutive temporary cells. The algorithm returns either 0 (false) or 1 (true).

```
temp0[-]temp1[-] >[-]+ >[-] <<
y[temp0+ temp1+ y-]temp0[y+ temp0-]
x[temp0+ x-]+
temp1[>-]> [< x- temp0[-] temp1>->]<+<
temp0[temp1- [>-]> [< x- temp0[-]+ temp1>->]<+< temp0-]
temp1[x+temp1[-]]
```

4.17 : x = x <= y

Attribution: Ian Kelly

x and y are unsigned. temp1 is the first of three consecutive temporary cells. The algorithm returns either 0 (false) or 1 (true).

```
temp0[-]temp1[-] >[-]+ >[-] <<
y[temp0+ temp1+ y-]
temp1[y+ temp1-]
x[temp1+ x-]
temp1[>-]> [< x+ temp0[-] temp1>->]<+<
temp0[temp1- [>-]> [< x+ temp0[-]+ temp1>->]<+< temp0-]
```

4.18 : z = x > y

Attribution: User:ais523

This uses balanced loops only, and requires a wrapping implementation (and will be very slow with large numbers of bits, although the number of bits otherwise doesn't matter.) The temporaries and x are left at 0; y is set to y−x. (You could make a temporary copy of x via using another temporary that's incremented during the loop.)

```
temp0[-]temp1[-]z[-]
x[ temp0+
y[- temp0[-] temp1+ y]
temp0[- z+ temp0]
temp1[- y+ temp1]
y- x- ]
```

4.19 : x = not x (boolean, logical)

Attribution: Jeffry Johnston

The algorithm returns either 0 (false) or 1 (true).

```
temp0[-]
x[temp0+x[-]]+
temp0[x-temp0-]
```

Attribution: Sunjay Varma

Another version for when you can consume x (mutate its value). Also assumes that x is either 0 or 1. If you do not want to consume x, you can still use this algorithm. Just copy x to another cell, then apply the operation. The algorithm returns either 0 (false) or 1 (true).

```
temp0[-]+
x[-temp0-]temp0[x+temp0-]
```

Modification of Sunjay's version above

```
temp0[-]+
x[[-]temp0-x]temp0[-x+temp0]
```

Then there's undoubtedly no mistake. When x>1, the result will be as same as x=1 .

Attribution: Yuval Meshorer

Even another version that consumes x. Returns 0 (false) if x is 1 (true) and 1 if x is 0.

```
temp0[-]
x[temp0-x-]temp0+
```

Attribution: FSHelix

A harder-to-understand version that actually is "y = not x", which preserves x but needs 3 continuous cells in total.

Maybe using it for calculating "y = not x" is not necessary, but I think this idea will be quite useful in some cases.

In fact the idea is also embodied in other codes in this page.

```
#Define these 3 cells as x, y=1 and t=0.

x>y[-]+>t[-]<<x
[>y[-]]>[>t]
```

According to whether x==0 or not, there are two different run modes because the position of the pointer changes in the "[]" loop.

The following is the process of the second line, "*" means the pointer is here.

```
If x==0:                                   If x!=0:
                      x   y   t                                    x   y   t
                     *0   1   0                                   *1   1   0
[>y[-]]              *0   1   0            [>y[-]]                 1  *0   0
[>y[-]]>              0  *1   0            [>y[-]]>                1   0  *0
[>y[-]]>[>t]          0   1  *0            [>y[-]]>[>t]            1   0  *0
```

Attribution:User:A It is based on x=x==y, where I set y into 0.

```
y+x[-y-x]+y[x-y[-]]x
```

4.20 : x = x and y (boolean, logical)

Attribution: Jeffry Johnston

The algorithm returns either 0 (false) or 1 (true).

```
temp0[-]
temp1[-]
x[temp1+x-]
temp1[
temp1[-]
y[temp1+temp0+y-]
temp0[y+temp0-]
temp1[x+temp1[-]]
]
```

Attribution: Sunjay Varma

Consumes x and y (leaves them as zero at the end of the algorithm) and stores the result in z. For short-circuit evaluation, don't evaluate x or y until just before they are used.

The algorithm returns either 0 (false) or 1 (true).

```
z[-]
x[
y[z+y-]
x-
]
y[-]
```

Attribution: Yuval Meshorer

Consumes x and y and outputs 1 in z if both x and y are 1, else 0.

```
z[-]
x[
-y[-z+y]
x]
```

INPUT			OUTPUT		
x	y		x	y	z
0	0		0	0	0
0	1		0	0	0
1	0		0	0	0
1	1		0	0	1

4.21 : z = x nand y (boolean, logical)

Attribution: FSHelix

Consumes x and y and outputs 0 in z if both x and y are 1, else 1.

```
z[-]+
x[
y[z-y-]
x-
]
y[-]
```

INPUT			OUTPUT		
x	y		x	y	z
0	0		0	0	1
0	1		0	0	1
1	0		0	0	1
1	1		0	0	0

4.22 : x = x or y (boolean, logical)

Attribution: Jeffry Johnston

The algorithm returns either 0 (false) or 255 (true).

```
temp0[-]
temp1[-]
x[temp1+x-]
temp1[x-temp1[-]]
y[temp1+temp0+y-]temp0[y+temp0-]
temp1[x[-]-temp1[-]]
```

Attribution: Yuval Meshorer

Returns 1 (x = 1) if either x or y are 1 (0 otherwise) If you use it in the case that x>1 or y>1,please make sure it won't cause overflow problem.

For example,if x=1 and y=255, than x will be 0.

```
x[
y+x-]
y[
x+y[-]
]
```

Attribution: Sunjay Varma

Consumes x and y (leaves them as zero at the end of the algorithm) and stores the result in z. For short-circuit evaluation, don't evaluate x or y until just before they are used.

If you don't care about short-circuit evaluation, temp0 can be removed completely. If temp0 is removed and both x and y are 1, z will be 2, not 1. This is usually not a problem since it is still non-zero, but you should keep that in mind.

Or there's a way to fix it, add these codes to the end:

```
z[x+z[-]]
x[z+x-]
```

The algorithm returns either 0 (false) or 1 (true).

```
z[-]
temp0[-]+
x[
z+
temp0-
x-
]
temp0[-
y[
 z+
 y-
]
]
y[-]
```

Attribution: Yuval Meshorer

Consumes x and y, does not use a temporary cell. Makes z 1 (true) or 0 (false) if either x or y are one.

```
z[-]
x[y+x-]
y[[-]
z+y]
```

INPUT			OUTPUT			
x	y		x	y	z	temp0
0	0		0	0	0	0
0	1		0	0	1	0
1	0		0	0	1	0
1	1		0	0	1	0

4.23 : x = x nor y (boolean, logical)

Attribution: FSHelix

Consumes x and y and outputs 0 in x if both x and y are 1, else 1. Used an extra cell "z" to avoid the overflow problem like the one mentioned in x = x or y.

```
x[z+x[-]]
y[z+y[-]]
z[x+z[-]]
```

INPUT			OUTPUT	
x	y		x	y
0	0		0	1
0	1		0	0
1	0		0	0
1	1		0	0

4.24 : z = x xor y (boolean, logical)

Attribution: Yuval Meshorer

Consumes x and y. Makes z 1 (true) or 0 (false) if x does not equal y. Finishes at y.

```
z[-]
x[y-
x-]
y[z+
y[-]]
```

INPUT			OUTPUT		
x	y		x	y	z
0	0		0	0	0
0	1		0	0	1
1	0		0	0	1
1	1		0	0	0

4.25 : z = x xnor y (boolean, logical)

Attribution: FSHelix

Consumes x and y. Makes z 1 (true) or 0 (false) if x equal y. Finishes at y.

```
z[-]+
x[
 y-
 x-
]
y[
 z-
 y[-]
]
```

INPUT			OUTPUT		
x	y		x	y	z
0	0		0	0	1
0	1		0	0	0
1	0		0	0	0
1	1		0	0	1

4.26 : while (x) { code }

Attribution: Sunjay Varma

To implement a while loop, you need to evaluate the condition x both before the loop and at the end of the loop body.

```
evaluate x
x[
 code
 evaluate x again
 x
]
```

break and continue

Attribution: Sunjay Varma

To implement break and continue statements in loops, consider that the following two pieces of pseudocode are functionally equivalent:

```
while (foo) {
if (bar == foo) {
 if (x > 2) {
  break;
 }
 else {
  // do stuff
 }
 // do stuff
}
// update foo for the next iteration
}

// Equivalent without break statement:
while (foo) {
shouldBreak = false
if (bar == foo) {
 if (x > 2) {
  shouldBreak = true
 }
 else {
  // do stuff
 }
  // don't evaluate any more code in the loop after breaking
 if (!shouldBreak) {
  // do stuff
 }
}
if (shouldBreak) {
 // so that the loop stops
 foo = 0
}
else {
 // update foo for the next iteration
}
}
```

Notice that we need to guard all code after the break statement in the loop to prevent it from running. We don't need to guard in the else statement immediately after the break statement because that will never run after the break statement has run.

This approach allows us to implement break and continue statements in brainfuck despite the lack of sophisticated jump instructions. All we're doing is combining the concept of an if statement (defined below) with the while loop we just defined and applying it here.

Implementing a continue statement is the same thing except you never guard loop updating code:

```
while (foo) {
if (bar == foo) {
 if (x > 2) {
  continue;
 }
 else {
  // do stuff
 }
 // do stuff
}
// update foo for the next iteration
}

// Equivalent without continue statement:
while (foo) {
shouldContinue = false
if (bar == foo) {
 if (x > 2) {
  shouldContinue = true
 }
 else {
  // do stuff
 }
  // don't evaluate any more code in the loop after
continuing
 if (!shouldContinue) {
  // do stuff
 }
}
// This code stays the same after a continue because we still
want to move on to the next iteration of the loop
// update foo for the next iteration
}
```

To implement both break and continue, you can compose the concepts here and make any combination you want. You can consider break and continue statements to be "sugar" that needs to be "desugared" in your brainfuck code.

4.27 : if (x) { code }

```
temp0[-]
temp1[-]
x[temp0+temp1+x-]temp0[x+temp0-]
temp1[
  code
temp1[-]]
```

or alternatively:

```
temp0[-]
x[
  code
  temp0
]x
```

or alternatively if you don't need x anymore:

```
x[
  code
  x[-]
]
```

4.28 : if (x) { code1 } else { code2 }

Attribution: Jeffry Johnston (39 OPs)

```
temp0[-]temp1[-]
x[temp0+temp1+x-]temp0[x+temp0-]+
temp1[
  code1
  temp0-
temp1[-]]
temp0[
  code2
temp0-]
```

Attribution: Daniel Marschall (33 OPs)

```
temp0[-]+
temp1[-]
x[
 code1
 temp0-
 x[temp1+x-]
]
temp1[x+temp1-]
temp0[
 code2
temp0-]
```

This is an alternate approach. It's more efficient since it doesn't require copying x, but it does require that temp0 and temp1 follow x consecutively in memory.

Attribution: Ben-Arba (25 OPs)

```
temp0[-]+
temp1[-]
x[
 code1
 x>-]>
[<
 code2
 x>->]<<
```

4.29 : if (x == 0) { code }

The code examples in the following section work for this: if (x) { code1 } else { code2 }. Just have "code1" be empty.

4.30 : Divmod algorithm

A clever algorithm to compute div and mod at the same time:

```
# >n 0 d
[->+>-[>+>>]>[+[-<+>]>+>>]<<<<<<]
# >0 n d-n%d n%d n/d
```

If one does not need to preserve n, use this variant:

```
# >n d
[->-[>+>>]>[+[-<+>]>+>>]<<<<<]
# >0 d-n%d n%d n/d
```

This algorithm doesn't work when the divisor is 0 or 1.

4.31 : Modulus algorithm

If we do not need to compute the quotient as well, the following approach is shorter than the divmod algorithm.

```
# 0 >n 0 d 0 0 0
[>+>->+<[>]>[<+>-]<<[<]>-]
# 0 >0 n d-n%d n%d 0 0
```

As an additional advantage, this algorithm works even if the divisor is 1.

If n doesn't have to be preserved, the following variant can be used instead.

```
# 0 >n d 0 0 0
[>->+<[>]>[<+>-]<<[<]>-]
# 0 >0 d-n%d n%d 0 0
```

5 – Sample problem

we will show you the problem of programming that can be solved by brainfuck languages easily, i.e. the problem from : SPOJ.com – SBSTR1

Statement :

Given two binary strings, A (of length 10) and B (of length 5), output **1** if B is a substring of A and **0** otherwise.

Input :

24 lines consisting of pairs of binary strings A and B separated by a single space.

Output :

The logical value of: 'B is a substring of A'.

Example :

First two lines of input:

```
1010110010 10110
1110111011 10011
```

First two lines of output:

```
1
0
```

Solution :

Recommended: Please try your approach on https://www.spoj.com/submit/SBSTR1/, before moving on to the solution.

The next page is the source code of solution, the purpose of this section is, we can say that this the problem can be solved **by used brainfuck.**

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

6 – References

[1] – El Brainfuck, Brainfuck editor & optimizing interpreter. https://copy.sh/brainfuck/. Accessed January 15, 2019.

[2] – Katie, BrainFuck Programming Tutorial. https://gist.github.com/roachhd/dce54bec8ba55fb17d3a. Accessed January 15, 2019.

[3] – Esolangs, Brainfuck algorithms. https://esolangs.org/wiki/Brainfuck_algorithms. Accessed January 16, 2019.

[4] – Wikipedia, Brainfuck. https://en.wikipedia.org/wiki/Brainfuck. Accessed January 17, 2019.

[5] – Fatih ERIKLI, Visualizer Brainfuck. https://fatiherikli.github.io/brainfuck-visualizer/. Accessed January 18, 2019.

[6] – SPOJ, Substring Check(Bug Funny). https://www.spoj.com/problems/SBSTR1/. Accessed 18, 2019.

[7] – Muppetlabs, Brainfuck An Eight-Instruction Turing-Complete Programming Language. http://www.muppetlabs.com/~breadbox/bf/. Accessed 19, 2019.

[8] – Gynvael, Brainfuck Compiler in 125 bytes. https://gynvael.coldwind.pl/n/bf_125_bytes. Accessed January 20, 2019.