The Complete BrainFrix Documentation

Ivan Patarčić and Filip Jerleković, 2020.

Chapter 1. The Basics

The BrainFrick interpreter is based on the BrainF### language, which at its base features an array of cells which can hold values, similar to variables in standard, non-abstract languages. Let’s imagine it with the following visualisation:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Now, this cell array can be manipulated by a multitude of commands you will find in the rest of the handbook. For now, let’s consider the following example:

```
+ (no. 1.)

> (no. 3.)

+
```

Check further documentation for explanation of no. 1. and no. 3. Anyway, this little code snippet (+>) will alter the first cell, then switch to the second cell, and then alter the second cell. So, in conclusion, the basic cell part features an array of integers, a pointer which you can alter with no. 1. And no. 3., which is the pointer to the cell which is currently being altered. This example right here will alter the aforementioned array and make it like so:

1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The BrainFrick interpreter as a whole has 19 commands, as opposed to the original BF language which had 8. The commands are the following:

+	**1**	**add 1 to current cell**
-	2	Subtract 1 from current cell
>	3	Add 1 to the pointer
<	4	Subtract one from the pointer
[	5	Loop start
]	6	If the current cell is not zero, goes back to the loop start
,	7	Inputs ASCII value to current cell
.	8	Outputs the char from cell ASCII value
_	9	Inputs an integer instead of an ASCII value
;	10	Outputs a new line
:	11	Outputs the current integer value
*	12	Changes the limit of cells, default is 16 bits, lowest is 8
~	13	Switches between fast add and slow add mode
‘	14	Prints cell position
“	15	Prints the whole array up to the current cell
%	16	Assign current cell value to the static cell
#	17	Assign static cell value to the current cell
/	18	Nullify current cell
&	19	Add the past cell value to the current cell

This set of commands is all you will need for BrainFrick programming. In the following few chapters you will learn how to use all of them.

Good luck.

Chapter 2. **ADD, SUBTRACT** and **BIND**

ADD, **SUBTRACT**, and **BIND** are the three commands used in the interpreter as **+**, **-**, and **&**. Let's for example take the following example:

```
+>+&->+
```

This example will **ADD** to the first cell, **MOVE_RIGHT** to the second and then **BIND** the two cells. By **BIND** the cells I mean the second cell will be its current value plus the previous cell value. The previous cell is the cell on the **CURRENT_CELL_VALUE**, but minus one. E.g., if the cell is second and **BIND** is used, the second cell will be the second cell plus the first cell. The **ADD** and **SUBTRACT** commands are pretty self-explanatory, they **ADD** one or **SUBTRACT** one from the current cell. If **EIGHTBIT** is true and the cell has exceeded 255, the cell value is set to zero. If **EIGHTBIT** is false, then the number can go up to 2^{32}. If **SLOWADD** is false, the complexity of every **BIND** will be the cell before the current cell. If **SLOWADD** is true, then the complexity of the **BIND** will always be 1.

Chapter 3. **SHIFT_RIGHT** and **SHIFT_LEFT**

SHIFT_RIGHT (>) and **SHIFT_LEFT** (<) move the cell pointer one right

or one left. Let's take the previous example once again:

```
+>+&->+
```

In here we see two **SHIFT_RIGHT** commands. Meaning that the cell pointer was initially set to 0, as is with any program. This means that at the end the pointer will be shifted twice, and at the end the cell being altered will be the one with the index 2, or the third. If the pointer is set below zero, the program will keep working. If the pointer is set below zero **AND** either **ADD, SUBTRACT, BIND, NULLIFY, STATIC_ASSIGN_CURRENT, CURRENT_ASSIGN_STATIC, DEBUG_ARRAY_OUTPUT, INTEGER_INPUT, ASCII_INPUT, INTEGER_OUTPUT, ASCII_OUTPUT** or **BFLOOP_END** is executed, the program will quit. Basically, if the cell is accessed the program will quit.