# DrBrainfuck – Documentation

Tommaso Rodolfo Masera  Claudio Maggioni

December 2018

## Contents

# 1 User Level

## 1.1 Brief Introduction to *Brainf*ck*

*Brainf*ck* is a programming language supposed to resemble a working Turing machine and it consists of only eight commands.

A program written in *Brainf*ck* makes use of sequences of these commands and said sequence might actually have other characters in between that are promptly ignored and treated as comments instead.

The way *Brainf*ck* works includes a program and an instruction pointer, an array of byte cells initialized to 0 as well as a movable data pointer, starting from the leftmost position, to address such cells with the given instructions. What's more *Brainf*ck* makes use of the ASCII encoding for inputs and outputs.

The eight commands *Brainf*ck* is based on are the following:

`>` : increments the data pointer to point the cell to the right;

`<` : decrements the data pointer to point the cell to the left;

`+` : increases by one the byte at the data pointer;

`-` : decreases by one the byte at the data pointer;

`.` : prints as output the byte at the data pointer;

`,` : asks for an input to store in the byte at the data pointer;

`[` : if the byte at the data pointer is zero, jumps forward to the command after the matching `]` command instead of advancing the instruction pointer to the next instruction;

`]` : if the byte at the data pointer is non-zero, jumps backward to the command before the matching `[` command instead of advancing the instruction pointer to the next instruction;

## 1.2 About the Interpreter

**The interpreter is written in Racket and was developed using DrRacket 7.0**

### 1.2.1 Running the Program

You have two different options to run the program: a GUI and a CLI.

For the GUI open the "`gui.rkt`" file from either the 'DrRacket' environment or the Racket CLI tool.

As for the CLI version of the program you should use the "`./cli.rkt`" command followed by your "`filename.bf`" *Brainf*ck* file that you want to execute.

### 1.2.2 Current Features

The current status of the project includes a fully functioning *Brainf*ck* interpreter with a GUI capable of displaying input and output of the program.

The GUI includes a live display of a Turing machine tape as the program runs.

The program is also supported via command line as well as allowing direct user input in the *Brainf*ck* program while it runs.

# 2 Developer Level

## 2.1 Interpreter Execution

### 2.1.1 Program State

The entire program revolves around the main struct defined as:

```
; A ProgState is a (prog-state tape dp output program ip) where:
; - tape: Tape
; - dp: DataPointer
; - tape-len: Nat
; - output: String
; - program: Program
; - ip: InstructionPointer
; - error: Option<String>
; Interpretation: the current state of execution of a brainf*ck program.
```

```
(struct prog-state (tape dp tape-len output program ip error)
```

And, likewise, each term in the struct has its own type definition:

```
; A Byte is an Int between 0 and 255
; Interpretation: a byte in decimal notation.

; A Tape is a NEList<Byte>
; Interpretation: a tape in brainf*ck's Turing machine.

; A DataPointer (DP) is a NonNegInt
; Interpretation: a data pointer in the Brainf*ck language in a tape.

; A Program is a String of:
; - ">" (tape-right)
; - "<" (tape-left)
; - "+" (add1)
; - "-" (sub1)
; - "." (out)
; - "," (in)
; - "[" (loop-start)
; - "]" (loop-end)
; Interpretation: the brainf*ck program.
```

```
; A InstructionPointer (IP) is a NonNegInt
; Interpretation: a pointer to the instruction to execute.

; An ErrorCode is one of:
; - 'error1 (Interp: negative tape position when <)
; - 'error2 (Interp: non-matching [)
; - 'error3 (Interp: non-matching ])
; Interpretation: an error code for the bf interpreter.
```

### 2.1.2 Execute Function

The main aspects of the `execute` function, other than executing the program, include:

- The world state previously defined as a program state
- An asynchronous function call to get the user input when required by the program
- A callback function call defined as "`done`" which is called when an instruction is executed and that returns `#false` when the program is at its last instruction.

### 2.1.3 Interpreter Execution

In order to parse the *Brainf*ck* instructions correctly and ignore all other characters in a *Brainf*ck* file, the execute function requires a `cond` to give a condition to each valid *Brainf*ck* character and call the right function.

List of helper functions for execute:

**`exec-tape-right`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the > instruction executed.

**`exec-tape-left`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the < instruction executed.

**`exec-add1`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the + instruction executed.

**`exec-sub1`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the − instruction executed.

**`exec-out`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the . instruction executed.

**`exec-loop-start`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the [ instruction executed.

**`exec-loop-end`**:
ProgState -> ProgState
Given a ProgState, returns a new ProgState with the ] instruction executed.

**`exec-in`**:
ProgState ((Byte -> _) -> _) (ProgState -> _) -> _
Given a ProgState, a function that takes a callback function requiring a Byte and a function which takes the new ProgState, calls done with the input provided by get-input (provided by the call to the callback given in get-input).