Brainfuck for the Commodore 64

Vintage Computing Carinthia
Documentation by Wil

Version 1.0
April 2022

Contents

1 Brainfuck - What, Why and How

Brainfuck is a computer programming language introduced in 1993 by Urban Mueller. At first glance, it seems to combine the slowness of BASIC with the difficulty of Assembler programming, but actually it is exceeding both languages in these aspects. It is far slower than BASIC and far more difficult than 6502 Assembly. So, welcome to Brainfuck!

This is the documentation for Brainfuck64 V1.0 by Vintage Computing Carinthia. It is not the first implementation of a Brainfuck interpreter, there were at least nice implementations by TheDreams in 2001[1] and by Marco64 in 2009[2]. But now with Brainfuck64 you have acquired a Brainfuck development system for your Commodore 64!

2 The Brainfuck Language

2.1 Memory structure

A Brainfuck program consists of any combination of Brainfuck commands and, in addition, further text which is ignored when executing the program. This way, comments cab be directly written into the code as long as the do not contain any Brainfuck commands. A value of 0 marks the end of the code.

The working memory is accessed and manipulated by a running Brainfuck program. The program can only access one memory cell at a time indicated by a pointer to the current memory cell. The working memory is initialized with zeros each time the program starts and the pointer is initially pointing to the first cell.

2.2 The eight elementary commands

The Brainfuck language only has eight commands, yes eight. Each command is represented by a single character. The eight commands are:

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

. prints out the character with the ASCII value at the current memory cell.

, reads a character from keyboard and puts ASCII code into the current memory cell.

+ increases the value at the current memory cell by 1. If the maximum cell value (typically 255) is exceeded the value wraps to 0.

---

1. Brainf... https://csdb.dk/release/?id=74814
2. Brainfuck Interpreter https://csdb.dk/release/?id=74303

- decreases the value at the current memory cell by 1. If the cell value already contains its minimum value (0), the value wraps to the maximum cell value (typically 255).

> increases the pointer to the current memory cell by 1

< decreases the pointer to the current memory cell by 1

[if the current memory cell contains 0, the program flow jumps forward after the corresponding] command. If the current memory cell contains any other number than 0, the program continues with the command right after the [.

] if the current memory cell contains any other number than 0, the program flow jumps backward after the corresponding [. If the current memory cell contains 0, the program continues with the command right after the].

2.3 Calculations

2.3.1 Setting a value

set current memory cell to 0: [-] loops until the value is 0. All working memory cells are initialized with 0, so this code piece is only necessary if a cell was used before for some calculation.

set current memory cell to X: If the memory cell is 0 before, a sequence of X +-commands does the trick. For example, +++++ sets a value of 5.

set current memory cell to X (compact approach): Larger numbers can be defined in a more nifty way with loops. For example, +++++[>+++++++++++++<-]> yields a cell with a value of 65, calculated by 5 times adding 13. This uses an extra memory sell, but is way shorter than 65 +-commands in a row. an even shorter definition can be found using the wrapping of values: ----[---->+<]>++ counts up to 63 while counting down in steps 4. After the loop, the value is increased two times to yield again 65.

For a list of constants being expressed in compact Brainfuck code, see Brainfuck constants [3].

2.3.2 Moving and Duplicating a value

Movind a value is done by decrementing the original value while incrementing the new value: [>+<-] The target memory cell must be 0 (or set to 0) for this algorithm to work.

---

3. Brainfuck constants - Esolang https://esolangs.org/wiki/Brainfuck_constants

[>+>+<<-] copies the value of the current memory cell into the next two cells, the original cell is set to 0 in the process. To set the original value in the first cell, one value needs to be shifted back by adding this code: >>[<<+>>-] After the loop, the memory pointer is pointing at the 0 after the values.

2.3.3 Adding and subtracting values

[>+<-] adds the value of the current cell to the value of the next cell.

[>-<-] subtracts the value of the current cell from the value of the next cell.

2.3.4 Multiplying values

Multiplication the current cell value with a constant X works by adding X repeatedly to the result while decrementing the first cell value: [>(X times +)<-]

Multiplying two cell values [>[>+>+<<-]>>[<<+>>-]<<<-] through iterative addition.

2.3.5 Division

If the division has a remainder of 0, the following loop works well to divide by X [>+<(X times -)]

Dividing two cell values with a possible remainder requires a bit more effort: [->>+<-[>>>]>[[<+>-]>+>>]<<<<<] divides the current cell value by the next cell value and puts remainder and division result into the two consecutive cells.

2.4 Printing text

To output a text, set the corresponding ASCII value in a memory cell and output it with .

Since . does not change the cell value, the value from the previous output can be used to create next value. Thus, the name of a great band [4] can be coded in a very compact way:

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

Outputting value 10 creates a carriage return.

2.5 Program control flow

2.5.1 Repeat X times

This is on of the most common structures in Brainfuck, we have already used it, among others, for the multiplication algorithm.

(X times +)[code in the loop -]

The following example outputs 10 exclamation marks:

4. perhaps you expected ----[----->+<]>++.++.+.-. ?

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

The ASCII value of the exclamation mark is generated using two memory cells. The sequence [-]<[-] after output clears the values for the next run and adjusts the cell pointer.

2.5.2 If non-zero

The [command does already most of the job. To leave the loop, a zero value must appear at the current cell when reaching the] If the original value is not needed afterwards, a [-] does the job:

[code to be executed conditionally [-]]

If the next cell is known to be empty, the [-] an be omitted which preserves the tested value.

[code to be executed conditionally >]<

2.5.3 If zero

For testing for zero, we set a flag in the next cell to 1, then test the first cell value for non-zero. If this is the case, the flag is reset. In other words, the flag is calculated as a logical NOT of the current cell value.

>+<[>-<[-]]>[code to be executed conditionally -]

In combination with the subtraction method, the if zero pattern can also be used to formulate an "if equal" condition.

2.5.4 If then ... else ...

The following code implements an if zero then ... else ... structure.

>+<[>-< here comes the else code[-]]>[- here comes the then code [-]]

Due to the nested loop, the else-part comes first in the code. The else code needs to end with the following memory cell being 0.

2.5.5 Switch to different cases based on a value

This can be implemented in a similar way as the if-then-else approach.

For example:

```
switch(X)
case 3:
 print 1
case 8:
 print 2
case 6:
 print 3
default
```

```
 print ?
endswitch
```

for efficient comparison, the values should be sorted from low to high:

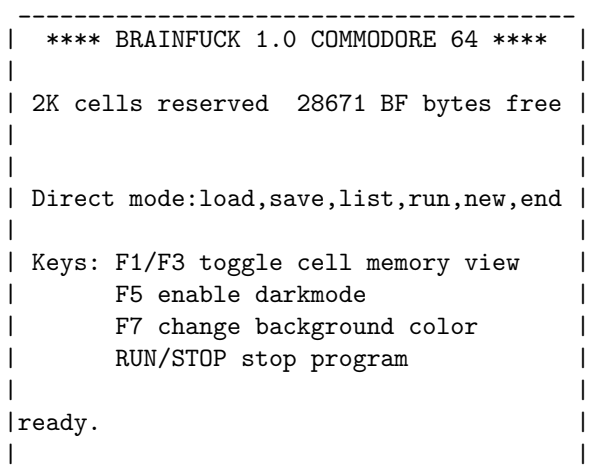
```
switch(X)
case 3:
 print 1
case 6:
 print 3
case 8:
 print 2
default
 print ?
endswitch
```

The respective Brainfuck code is then as follows:

```
>+<  set flag
---[ compare with 3
---[ compare with 6
--[  compare with 8
>-< default part
[-]----[---->+<]>.[-]< output print and clear used cell
[-]] leave loop
>[- case 8
-[----->+<]>--.[-]< [-] output 3 and clear mem cells
]<]  end case
>[- case 6
-[----->+<]>-.[-]< [-] output 2 and clear mem cells
]<]  end case
>[- case 3
-[----->+<]>.[-]< [-] output 1 and clear mem cells
]  end case and switch
```

3 Brainfuck on the Commodore 64

After loading and starting brainfuck64, the computer greets you with the following message:



There is a maximum of 2048 memory cells. Each cell is implemented as an 8-Bit integer number that can also wrap around. So if the current cell contains a value of 255, the + command will make it change to 0. In addition to the memory cells, there are 28 KB provided for Brainfuck programs.

3.1 The editor

The editor is a full screen editor using line numbers to add, delete or change the program.

To add a program line, simply type in a line number followed by the code to be stored in this number and press return. The maximum length of on line is 80 characters (two screen lines). If this line number already exists in your code, the line is replaced with the new line. To delete a line, enter the plain line number and press return. To delete the whole program, type NEW and press return.

Line numbers can be any integer number between 0 and 63999. It is recommended to assign line numbers with spacing to leave room for subsequent additions. A common practice is to use a spacing of 10, for example 10, 20, 30, 40, 50, ...

The computer sorts your lines according to their number into a program. To view your program, type LIST. For the case that your program is longer than one screen, LIST can be used together with a line number or a range of line numbers:

LIST	Shows the full program
LIST 100	List only line 100
LIST -100	List program from begin to line 100
LIST 100-	List program from line 100 to end of program
LIST 100-200	List program from line 100 to line 200

When executing a program, it makes no difference if a Brainfuck program is distributed over multiple lines or not. To start a program, type RUN and press return. RUN can also be used with a line number to start the program at the specified

number. During execution of the program, the border changes to a gray color. After the program has terminated the border color changes back. A running program can also be stopped by pressing the RUN/STOP key.

To leave Brainfuck and return to BASIC, type END and press return. Warning: a program in memory will be deleted in this case, so be sure to save your program before.

3.2 A first Brainfuck program

Start with an empty program and enter the following lines:

```
10 -[------->+<]>-.[-]
20 --[----->+<]>-.[-]
30 +[------->++<]>--.[-]
40 +[------->++<]>--.[-]
50 +[------->++<]>+.
```

Type LIST to check if all lines are there.
Type RUN and check the output.
Type RUN 20 and see what's different.
Type 30, press return and RUN again. What happened?
Type LIST 40, move the cursor to the line number and change it to 30, then press enter. Type LIST. Line 40 has now been copied into line 30.

Note that the functionality of the example program can also be written in a more compact manner by taking advantage of the value from the previous calculation for all characters but the first one.

```
1 -[------->+<]>-.-[->+++++<]>++.
2 +++++++..+++.
```

The execution of the compact version is also visibly faster.

3.3 Saving and loading Programs

3.3.1 Saving a program

Brainfuck programs are saved with the same commands as in BASIC. To save a program type

```
SAVE"programname",X
```

where X is the address of you peripheral storage device. For a disk drive, the number is usually 8.

If a file with the same name already exists, the SAVE command fails. There is a possibility to overwrite an existing file, however, be aware of a bug in the 1541 drive which eventually leads to a lost file. To avoid the bug, extend the filename with an optional drive number:

```
SAVE "@0:Name",8
```

3.3.2 Loading a program

To load a program, type

```
LOAD"programname",X
```

where X is the address of you peripheral storage device. For a disk drive, the number is usually 8. The program name can also use wildcards, where ? replaces an arbitrary character and * indicates an arbitrary number of characters. If a filename specified with wildcard characters matches multiple files on a disk, the first matching file is loaded.

3.3.3 Viewing the disk's directory

Disk drives allow to load a special directory file named $ containing a list of all files on the disk. Careful, loading the directory replaces the current program in memory. After loading with

```
LOAD"$",8
```

the directory can be viewed with LIST.

3.4 Character encoding

The Commodore C64 has a different implementation from standard ASCII, called the PET Standard Code of Information Interchange (PETSCII). To ensure compatibility, Brainfuck64 is converting the ASCII codes to their respective PETSCII representations. So a number of 65 will produce a big A, as it is intended in ASCII and the carriage return code is 10.

Apart from the ASCII standard codes, Brainfuck64 also interpretes PETSCII control codes and graphic characters. So it is possible to clear the screen, change the cursor color or put graphical characters on the screen.

A few codes are listed in the following table:

19	Cursor home position
135	F5 key (set black background)
136	F7 key (next background color)
17	Move cursor down
157	Move cursor left
29	Move cursor right
145	Move cursor up
18	Reverse mode on
144	Set black cursor color
31	Set blue cursor color
149	Set brown cursor color
159	Set cyan cursor color
151	Set dark gray cursor color
30	Set green cursor color
154	Set light blue cursor color
155	Set light gray cursor color
153	Set light green cursor color
150	Set light red cursor color
152	Set mid gray cursor color
129	Set orange cursor color
156	Set purple cursor color
28	Set red cursor color
5	Set white cursor color
158	Set yellow cursor color
32	Space
14	Switch font to upper/lowercase
142	Switch font to uppercase/graphics mode

3.5 Debugging

3.5.1 Viewing the working memory

Brainfuck64 supports a function for viewing the working memory during execution. Press F1 to switch to working memory view. This shows the first 1000 bytes of the working memory with life updates. The memory contents are depicted using hexadecimal numbers. The second memory page can be shown by pressing F1 a second time, the border color changes from black to dark gray when showing the second page. To leave the memory view, press F3.

3.5.2 Pause execution

To see if the code reaches a certain line and to pause there in order to check the memory state with F1, the following code can be used:

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

This works as long as the memory 5 cells ahead is not used, other wise the numbe of > and < needs to be increased. It is recommended to keep this code in a separate line and copy this line in between your program lines if you need it to stop. The code changes the background and prompts an input, then moves the memory pointer back to where it was.

3.5.3 Commenting out code parts

Make memory pointer point to a cell with 0 and put a [and] around the code to be commented out. Then the interpreter will skip this part.

```
[-][ this code will never be executed ]
```

Note that this has the side effect of setting the current memory cell to 0.

As an alternative, putting a [code] bracket after a] is guaranteed to be skipped, because the current memory cell must be 0 after exiting a [ ] loop.

4 Code examples on disk

4.1 Hello World

The program calculates the ASCII values for the the string "Hello World!" and outputs it to the screen.

4.2 Echo

The program reads from the keyboard and echo the input back on the screen. The program ends upon pressing the return key.

4.3 Brain - a PETSCII graphic

This program generates the graphic output of Logiker's PETSCII art of a brain with some graffiti below. The program outputs a character with code 142 in order to switch the charset on a Commodore computer. Thus, the output of this program would look different on other Brainfuck systems.

4.4 e - Euler's number

The program computes Euler's number, digit for digit. If you don't know this number, go and run the program. This program was written by Daniel Cristofani from brainfuck.org and was released under a Creative Commons BY-SA 4.0 license.

4.5 Life - Game of Life

The program simulates a Game of Life system within a wrapping 10x10 grid. Game of Life is a two-dimensional cellular automaton invented by John Conway. Each cell can have a dead or an alive state. The eight cells around a cell are the neighborhood of a cell. In each simulation step, cells change their state according to the following rules:

- an alive cell with less than 3 neighbors dies of loneliness
- a dead cell with exactly 3 alive neighbors is born (changes to alive state)
- an alive cell with more than 3 neighbors dies of overpopulation
- all other cells keep their state

To use the program, start it and wait until the (emtpy) grid is drawn. At the prompt you can either enter the coordinates of a cell to set or press enter to advance the simulation by one step.

Some patterns to start with:

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

The block is a static pattern, the bar oscillates and the pattern on the right is a glider that moves diagonally over the screen by making a copy of itself in two steps.

This program was written by Daniel Cristofani from brainfuck.org and was released under a Creative Commons BY-SA 4.0 license. We have added a . after the , in line 235 to make the input echo on the screen.

4.6 TTT - Tic-Tac-Toe

This program plays Tic-Tac-Toe. After starting the program makes its first move. The player can then respond by choosing a remaining field. Since Game of Life is actually a simulation, this is the only game in our examples.

The game does a lot of calculations and, therefore, takes a long time to respond, especially after the player enters his first move. If you are tired of waiting, press the RUN/STOP to stop the program.

This program was written by Daniel Cristofani from brainfuck.org and was released under a Creative Commons BY-SA 4.0 license. We have added a . after the , two times in line 15 to make the input echo on the screen.

4.7 Clocktower

This program switches to graphics mode and prints an F5-keycode to switch the background to black. Then a screen-sized graphic of the clocktower of Graz by night is printed. Greetings to Commodore Meetings Graz at this point! The output is using the PETSCII graphic characters and codes for changing the cursor color. Thus, the output of this program would look much different on other Brainfuck systems.

4.8 Primes

This program prints the prime numbers up to given number. After starting the program, it asks for "Primes up to". After entering a number the prime numbers up to the entered number are calculated and printed. Entering a number larger than 255 clips the number as an 8 bit number, so entering 258 is the same as entering 3.

This program was written by Daniel Cristofani from brainfuck.org and was released under a Creative Commons BY-SA 4.0 license. We have added a . after the , in line 25 to make the input echo on the screen.

4.9 Fantasie - A poem against war

Fantasie von Uebermorgen is a poem from German poet Erich Kaestner making a statement against war and that women would disallow their men to go to war. Kaestner published the poem in 1929, after The Great War, but before World War II. At the time of the release of this Version of Brainfuck64, there are still wars ongoing, even in Europe. This is terrible and incredible! We have added this poem to express our hope to end war some time. For good.