Seminar Report

brainfuck - Programming Language

Simon Mathis
simon.mathis@student.uibk.ac.at

18 February 2011

Supervisor: Dr. Georg Moser

Abstract

This paper gives a short introduction to the esoteric Programming Language brainfuck. It's aim is to point out the history and structure of brainfuck and gives a short comparison to Java. It will also point out what way you have to programme in brainfuck and what use it has nowadays.

Contents

1 Introduction

This paper is about brainfuck and starts with the history of the language itself. At first P' is described, the formal parent language which was created by Corrado Böhm (section 2.1), and after that talks about the developer of brainfuck Urban Müller (section 2.2) and his intentions and goals when he started to develop brainfuck. After the history there will be a section about the commands and the syntax of brainfuck (section 3) and some examples to point out how to programme in this language. It also contains a short sketch why it is Turing complete (section 5) and give a short comparison to Java(6). In the conclusion(section 7) i will talk about the way it was getting to know brainfuck and why it's not useless to know about it.

2 History of brainfuck

2.1 Corrado Böhm - P'

The formal parent language P' of brainfuck was designed in 1964 by Corrado Böhm, who was professor at the University of Rome. P' was introduced in his article "On a family of Turing machines and the related programming languages"[5]. It was the first language that was proved Turing complete, that did not contain a statement to jump to some arbitrary location like GOTO. P' was a totally formal language and there was no compiler written for it. Except of the two commands for input and output it had the same syntax as brainfuck, using the six symbols +,-, <, >,[and].

2.2 Urban Müller - brainfuck

Urban Müller is a swissmen who worked for Amiga before he designed brainfuck. He also developed the Softwarepackage Aminet[1] which was at that time the largest package for Amiga, containing multimedia librarys for movies music and pictures. In 1993 Urban Müller designed the esoteric programming language called brainfuck.[4] The language itself provides 8 commands for the programmer, and each programm consists of this 8 commands, every other symbol is ignored. The commands are the symbols from Corrado Böhms P' and , for input and . for output.
The intention of Urban Müller was at first design an own language. His targets were to create a language which is difficult to read and hard to understand even if someone is used to read programmes. Another goal he wanted to reach is a smallest possible compiler, inspired by the programming language FALSE with its 1024 bytes compiler.[4] The first compiler Urban wrote had 296 bytes, which is very small. And the challenge to read brainfuck programmes is huge if you look at the examples in section 4.2. Nowadays there are brainfuck compilers for nearly every operating system like brainfucked[2] for Windows which has 799

[1] Aminet.net
[2] http://home.arcor.de/partusch/html_de/bfd.html

byte. There is also a compiler for x86 Linux which has only 171 bytes[3], and an interpreter for MS-Dos with 98 Bytes written by Bertram Felgenhauer.[4]

3 Commands and Syntax

In this section some command examples are presented for easier understanding of brainfuck. And give a sketch about the basic concept of brainfuck.

3.1 Concept

The langauge brainfuck consists of an array of bytes (at least 30 000) which are all initialized with zero. And a pointer standing at the first byte of the array, an input and an outputstream.

Definition 3.1. The following is the grammar of instructions of P'
`I ::= Xi`
`Xi:= +` increments the byte at the pointer by one
`Xi:= -` decrements the byte at the pointer by one
`Xi:=` < decrements the pointer
`Xi:=` > increments the pointer
`Xi:= "["` if the current byte is zero jump to the next "]"
`Xi:= "]"` if the current byte is non zero jump to the matching "["

The language brainfuck itself is a trivial extension of P' (3.1) for the input and output of the values in the array with the commands . for output and , for input.

3.2 Addition

Example 3.2.

```
0 ,>++++++[<−−−−−−−−>−]
1 ,[<+>−]<.
```

If you want to add two values in brainfuck you first have to substract 48 (line 0) from one of them because the ASCII value of the input is 48 greater than the real number. After that you have to move the first number onto the second and print it.(line 1)

3.3 Copying a Byte

Example 3.3.

```
0 >[−]<
1 [−>+<]
```

[3] `http://gilesbowkett.blogspot.com/2007/05/brainfuck-compiler-for-linux.html`
[4] `http://de.wikipedia.org/wiki/Brainfuck`

If you want to copy a byte you have to start with clearing the target byte(line 0). After the clearing just move the byte to the target byte.(line 1) If you move a byte you lose the original source, but you can move it to two different locations at the same time.

4 Examples

In the Examples section there will be two examples explained, on the one hand "Hello World!" and on the other "99 bottles of beer" which was the topic of the specialisation seminar.

4.1 Hello World

4.1.1 Without Comments

Now a small example for a programme in brainfuck. Probably the most famous programme in computer history, "Hello World!". Here the programme is shown without comments, to point out that as mentioned in 2.2 the language is really hard to read.

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

4.1.2 With Comments

Hello World in brainfuck looks not very easy to understand so that is the version with comments:

Example 4.1.

```
++++++ ++++++
initialize counter (cell #0) to 10
[
use loop to set the next four cells to 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'
```

As brainfuck only can display letters with their ASCII values in a byte on his array, we have to reach the values somehow. For this we use the first loop adding 70/100/30/10 to the bytes, because 70 is near the value for uppercase letters, 100 near the lowercase letters, 30 near the shout sign and 10 near the newline.

4.2 99 bottles of beer

The 99 bottles of beer programme described was written by Aki Rossi[7]. The first thing in the 99 bottles of beer programme that you have to do is create output registers. And after it display the 99 bottles of beer on the wall line. I will try to explain the code of the 99 bottles of beer section per section. At the beginning of the code the beer counter in register 9 is set to 99, with the loop in line 01.

```
00 >>>>>>>>>
01 >++++++++++[-<++++++++++>]<-
02 <<<<<<<<<<
```

After that some output registers are created, for the comma in register, SP in register and LF in register. And the pointer is moved to the beercounter again with line 09. At first there is 40 added to register 1-4 in line 03 and 64 to register 3 and 4 in line 04 to reach the ASCII values faster as creating each one seperate. The ASCII values in the output registers are used to reach the letters that we later need faster. After that we set register 0 to Linefeed in line 06, register 1 to SP in line 07 and register 2 to comma in line 08.

```
03 ++++++++++[->++++>++++>++++>++++<<<<]
04 ++++++++[->>>++++++++>++++++++<<<<]
05 ++++[->>>>++++<<<<]
06 ++++++++++
07 >----------
08 >++++
09 >>>>>>>
```

And then we can start the loop for the verse itself in line 10. And initialize 3 states for repeating lines in registers 5-7 with lines 12-14.

```
10 [
11 <<<<
12 +++
13 >+
14 >++
15 <<
```

After that the "N bottles of beer" loop in line 16-25 is started. For the start we move to register 9 and go into the loop if we have bottles left on the wall in

line 18 and print the number with the conversion routine in the loop from line 19 to 24.

```
16  [
17 >>>>
18  [
19 [->+>+<<]>>[-<<+>>]<[>++++++++++[->>+>+<<<]
20 <[>>>[-<<<-[>]>>>>[<[>]>[----------->>]<++++
21 ++[-<+++++++++>]<<[<->[->-<]]>->>>[>]+[<]<<[
22 ->>>[>]<+[<]<<]<>>]<<]<+>>[->+<<+>]>[-<+>]<
23 <<<<]>>[-<<+>>]<<]>[-]>>>>>>>[>]<[.[-]<]
24 <<<<<<<
25  ]
```

After we have the number we need to find out if we need "no bottle of beer" only "bottle of beer" or "bottles of beer". We increment register 11 and the switch to register 9 with line 26 and if it's is empty we go to register 11 which has a value and write "no" in line 30, else we go to register 12 which is zero and skip the loop in line 28-34.

```
26 >+<<
27  [>]>>
28  [
29 <<<<<<<
30 ++++++.+.
31 -------
32 >>>>>>>
33 >>
34  ]
```

Then we print "bottles of beer", which is just swichting between the registers (0) and (4) and changing their values. In the lines 37 to 41 we write the word "bottle" and after that check if we need "bottles" or "bottle" in (12) [43] and if it's needed ((12) is not empty) we go into the loop [44-48]. Then we need a SP [50] and "of" [51-52] and "beer"[54-56] and reset all the registers used [57].

```
35  <[-]<[-]<
36 <<<<<<<<.
37 >>------.
38 ++++++++++++.
39 >----..
40 <---.
41 -------.
42 >>>>>>->>>+<<<
43  [>]>>
44  [
45 <<<<<<<<<
46 -.+
47 >>>>>>>
48  ]
```

```
49 >-<<<+<<<<<
50 <<<.
51 >>>-----.
52 <+.
53 <<.
54 >>----.
55 +++..
56 >+++.
57 <+++>++++++
58 >>
```

After the bottles we need "on the wall" that contains SP in line 60,62 and 65 "on"in line 61 "the"in line 63 and 64 "wall" in line 66-68. A reset is needed after every loop which is done in line 69. With line 72 and 73 we write comma and LF.

```
59 [
60 <<<<<.
61 >>+++++++.-.
62 <<.
63 >>>----.
64 <------.---.
65 <<.
66 >>>+++.
67 <----.
68 ++++++++++..
69 ----->+>>
70 -
71 ]
72 <<<<.
73 <<.
```

Then we start with "take one down". Where we have to print the letters one after another. If the skipmark, register 7, is not blank we skip the loop in line 77 to 109 and are at the end, else we go into the loop to write "take one down and pass it around". Also we have to decrease the beercounter, in register 9, with line 78 if we take one bottle down. After that we go to register 4 with line 79 to start with the writing. We start with "take" reached from register 4 and register 3 with line 80-83 and go on with "one" near from register 4 in line 85 and "down" reached from register 3 and 4 with line 87-89.

```
74 >>>>>>>
75 -
76 [>]>>
77 [
78 -
79 <<<<<
80 ----.
81 <-------.
```

```
82 >−−−−−−−−−.
83 <++++.
84 <<.
85 >>>++++.−.<.
86 <<.
87 >>−.>+.
88 ++++++++.
89 −−−−−−−−−.
90 <<<.
```

And the last thing is "and pass it around". Which is just the same technique like above. With the words "and" in line 91, "pass" in line 93 and 94 "it" in line 96 and "around" in line 98-102. After the words we again reset the registers in line 103, write comma in line 104 LF and in line 105. Then we increment register 6 with line 107 and go to register 10 with line 108.

```
 91  >>−−−.>.<+++.
 92  <<.
 93  >>>++.<−−−.
 94  >+++..
 95  <<<.
 96  >>++++++++.>+.
 97  <<<.
 98  >>−−−−−−−−.
 99  >−−.−−−.
100 ++++++.
101 −−−−−−−.
102 <+++.
103 ++++>++++++++++
104 <<.
105 <<.
106 >>−−−−−−−−−−
107 >>>>+
108 >>>>
109 ]
```

After the whole "N bottles of beer on the wall, take one down and pass it around" we need to restart it. We switch to register 5 decrease it with line 115 and jump up to line 18 if it's not empty. If it is emtpy we increment register 7, write LF reset the comma with line 116 and go back to the beer counter in register 9 with line 117. If it's is zero we are at the end in line 118, else we go to line 16.

```
110 <<<
111 <<
112 −
113 ]
114 >>+
115 <<<<<<<.
```

```
116 >>++++++++++++
117 >>>>>>>
118 ]
```

That was the programme for 99 bottles of beer on the wall. It's quite difficult and not very easy to understand, without comments it would take a long time, and a lot of paper to understand what the programme is doing.

5 Turing Completeness

In this section there will be a proof that brainfuck is Turing complete and a description of the so called counter machine. To proof that brainfuck is Turing complete I will start with the definition of Turing completeness itself.

Definition 5.1. A machine is called Turing complete if it's able to every Turing computable function. Or in other words, a machine that is able to simulate a universal Turing machine.

Now I will give a short sketch about how it's possible to proof that brainfuck is Turing complete. At first I need the definition of the counter machine. Which is proofed to be Turing complete[6].

Definition 5.2. A counter machine programme has as storage a finite number of counters(also called registers or cells) X0,X1,X2... , each holding a natural number. The instructions allow testing a counter for zero or incrementing or decrementing a counters content (0-1 = 0, i-1 = i-1). Every counter is initialized with zero except for the input.
The following is the grammer of CM instructions I.

```
I ::= Xi
Xi:= Xi+1
Xi:= Xi-1
Xi:= if Xi=0 goto l else l'
```

A short example for a counter machine programme which adds two values in counter 1 and counter 2. It's easy to compare it with line 1in the Example 5.4 at the bottom . Line 0 of this example is just because of the ASCII values used in brainfuck, but line 1 is just like this programme, just shifting the one value onto the other.

Example 5.3.
```
1 if X2 =0 then goto 5 else 2 ;
2 X2:=X2-1;
3 X1:=X1+1;
4 if X0 = 0 then goto 1 else 5;
5 HALT
```

For comparing the addition example in brainfuck again.

Example 5.4.

```
0 ,>++++++[<--------->-]
1 ,[<+>-]<.
```

Theorem 5.5. *The Programming language brainfuck is Turing complete.*

Proof. With the counter machine we have a related machine with programmes that is proved Turing complete. If it's possible to map every brainfuck programme into a counter machine programme and vice versa it's proved that brainfuck also is Turing complete. For this we use the grammar from 5.2 and rewrite it with equivalent instructions from brainfuck.

	Counter machine	brainfuck
I	Xi	Xi
Xi	Xi+1	+
Xi	Xi-1	-
Xi	if Xi=0 goto l else l'	[Xi]
Xi	Xi Xi	Xi Xi

With this grammar we can map every brainfuck programme into an equivalent counter machine programme, so we showed that it is Turing complete. □

6 Comparison - brainfuck and Java

In this section brainfuck will be compared with Java[2], one of the most used programming languages today. And the advantages of brainfuck will be pointed out.

6.1 Java

Java[2] is one of the most used programming languages in the world. It's principle is object oriented and there is a huge number of libraries implemented for Java for using devices, handling data and many more cases. Java code is compiled into bytecode that is running on the so called Java Virtual Machine, which can be installed on nearly all operating systems.

6.2 Comparison

It is not very easy to compare Java to brainfuck, because the programming languages are that different, but I will try to point out one advantage of brainfuck to Java, the size of the programmes and the compiler itself.

For a little example the Hello World in Java:

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

The size of the compiler of brainfuck and Java differs quite a lot, and the programmes also are different. This table shows with 2 programmes and the compiler, what difference there is in case of required memory between Java and brainfuck.

	brainfuck	Java
Compiler	98 byte	min 39 kbyte
Hello world	113 byte	130 byte
B. Beer	976 bytes	3,4 kbyte
sum	1187 bytes	42,4 kbyte

In this tabular it's clear that brainfuck does not need huge memoryspace, and with the new smartphones it's possible to compile on them. There are for example compilers to compile on Palm[3] and Android[1]. It's theoretically possible to compile Java directly on a smartphone, but with the limited resources it is not practicable.

7 Conclusion

When I chose brainfuck as my topic in the specialisation seminar I didn't know very much about it, despite that it was difficult to read, because I once saw a t-shirt with a brainfuck code on it. But after I started to read a little about it, and try some very easy examples by myself, I started to feel challenged about the minimization of a programme and it was very funny to think about some loops, and making them shorter. The history of brainfuck was also very interesting, to get to know about Corrado Böhm and Urban Müller and the origin of brainfuck. After all it was a nice topic to choose, and i had fun to get to know about all the unusual things brainfuck has.

References

[1] Android homepage. `www.android.com`.

[2] Java homepage. `http://www.java.com`.

[3] Palm homepage. `www.palm.com/de/de/index.html`.

[4] *Esoteric Programming Languages*. Books LLC, 2010.

[5] C. Boehm. On a family of turing machines and the related programming language. *ICC Bulletin*, 3:187–194, 1964.

[6] N. D. Jones. *Computability and Complexity*. MIT Press, 1997.

[7] A. Rossi. 99 bottles of beer in brainfuck. `http://99-bottles-of-beer.net/language-brainfuck-1539.html`.