Brainfuck

1 Brainfuck

1.1 Brainfuck — Geschichte und Umfeld

Brainfuck ist eine sogenannte *esoterische Programmiersprache* . Sie wurde 1993 vom Schweizer Urban Müller entworfen mit dem Ziel, eine Sprache mit möglichst kleinem Compiler zu erschaffen. Die Programmiersprache Brainfuck besteht aus einem Befehlssatz von genau acht Zeichen, nämlich

$$< \quad > \quad + \quad - \quad [\quad] \quad . \quad \text{und} \quad ,$$

Obwohl es auf den ersten Blick unglaublich erscheint, kann in dieser Programmiersprache prinzipiell jede Funktion (d.h. jedes Ein-Ausgabe-Verhalten von Bitmustern) berechnet werden, die auch mit einer anderen Programmiersprache wie Java oder C++ berechnet werden kann. Man nennt solche Sprachen Turing-vollständig.

1.2 Esoterische Programmiersprachen

Wikipedia definiert esoterische Programmiersprachen wie folgt:

> Esoterische Programmiersprachen sind Programmiersprachen, die nicht für den praktischen Einsatz entwickelt wurden, sondern ungewöhnliche Sprachkonzepte umsetzen. Eine einfache Bedienung ist selten, teilweise werden Sprachen konzipiert, um möglichst komplizierte Algorithmen oder unverständliche Syntax zu haben, oft aber auch um neue Ideen auszuprobieren, oder um ungewöhnliche Möglichkeiten wie extreme Vereinfachung aufzuzeigen. Mit Esoterik selbst haben „esoterische Programmiersprachen“ nichts zu tun, der Begriff greift lediglich die vermeintliche Ablehnung der Rationalität im esoterischen Kontext auf.[1]

Ein interessanter Beitrag zum Thema esoterische Programmiersprachen ist im Artikel „Hexenwerk — Ein Plädoyer für esoterische Programmiersprachen“ von Oliver Lau in der c't 22/07, S. 192-199 zu finden. Eine fast unerschöpfliche Quelle zum Thema ist das Esolang Wiki unter `http://esoteric.voxelperfect.net/wiki/Main_Page`. Seien Sie aber gewarnt: Man kann sich in diesen Spielerein und Spinnereien sehr schnell verlieren und verlieben — beschweren Sie sich also nicht bei mir, wenn Sie versuchen einen C++ Compiler in Whitespace zu schreiben.

[1] `http://de.wikipedia.org/wiki/Esoterische_Programmiersprache`

1.3 Brainfuck — Das Konzept

Stellen Sie sich ein (unendlich) langes Band vor, dass in einzelne Felder aufgeteilt ist. In jedem Feld steht zu Beginn eine Null. Stellen Sie sich nun vor, dass es einen Schreib- und Lesekopf gibt, der sich über das Band (jeweils von Feld zu Feld, in beide Richtungen) bewegen und folgende Aktionen ausführen kann:

- die Zahl, die im aktuellen Feld steht, auf einem Bildschirm ausgeben.
- eine Zahl von einer Tastatur einlesen und diese ins aktuelle Feld schreiben (und damit den vorherigen Eintrag ersetzen).
- die Zahl im aktuellen Feld um Eins erhöhen, d.h. wenn x im aktuellen Feld steht, wird dies durch den Wert $x + 1$ ersetzt.
- die Zahl im aktuellen Feld um Eins vermindern, d.h. wenn x im aktuellen Feld steht, wird dies durch den Wert $x - 1$ ersetzt.

Um einer solchen Maschine Befehle zu erteilen, stehen in Brainfuck folgende Befehle zur Verfügung

`>` Gehe einen Schritt nach rechts ins nächste Feld.

`<` Gehe einen Schritt nach links ins nächste Feld.

`.` Gib die Zahl im aktuellen Feld (als ASCII-Zeichen, siehe Tabelle 1) auf dem Bildschirm aus.

`,` Lies ein Zeichen von der Tastatur und schreibe dessen ASCII-Wert ins aktuelle Feld.

`+` Erhöhe die Zahl im aktuellen Feld um Eins.

`-` Vermindere die Zahl im aktuellen Feld um Eins.

Beispiel 1. Das folgende Programm liest also ein Zeichen von der Tastatur ein, speichert dessen ASCII-Wert im aktuellen Feld, erhöht den Wert im aktuellen Feld um Eins und gibt dann den Wert im aktuellen Feld wieder aus.

```
,+.
```

Gibt man z.B. das Zeichen ‚a‘ ein, dessen ASCII-Wert 97 ist, wird das Zeichen mit dem ASCII-Wert 98 ausgegeben, was dem Zeichen ‚b‘ entspricht. Der Online Brainfuck Interpreter unter `http://koti.mbnet.fi/villes/php/bf.php` ergibt folgenden Debug-Output:

```
0 (0): (The program contains 3 instructions.)

1 (0): , | read in a (97)
2 (1): + | a[0]= 98
3 (2): . | output '98' b
```

Nr.	Symbol	Nr.	Symbol	Nr.	Symbol	Nr.	Symbol	Nr.	Symbol
33	!	34	"	35	#	36	$	37	%
38	&	39	’	40	(	41	)	42	*
43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4
53	5	54	6	55	7	56	8	57	9
58	:	59	;	60	<	61	=	62	>
63	?	64	@	65	A	66	B	67	C
68	D	69	E	70	F	71	G	72	H
73	I	74	J	75	K	76	L	77	M
78	N	79	O	80	P	81	Q	82	R
83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\
93	]	94	^	95	_	96	‘	97	a
98	b	99	c	100	d	101	e	102	f
103	g	104	h	105	i	106	j	107	k
108	l	109	m	110	n	111	o	112	p
113	q	114	r	115	s	116	t	117	u
118	v	119	w	120	x	121	y	122	z
123	{	124	|	125	}				

Tabelle 1: Liste der gebräuchlichsten ASCII-Zeichen

Beispiel 2. Das folgende Programm liest drei Zeichen ein und gibt diese in umgekehrter Reihenfolge wieder.

```
,>,>,.<.<.
```

Gibt man z.B. ‚abc‘ ein erhält man als Output ‚cba‘ Der Online Brainfuck Interpreter unter `http://koti.mbnet.fi/villes/php/bf.php` ergibt folgenden Debug-Output:

```
0 (0): (The program contains 10 instructions.)

1 (0): , | read in a (97)
2 (1): > | array pos. now 1
3 (2): , | read in b (98)
4 (3): > | array pos. now 2
5 (4): , | read in c (99)
6 (5): . | output '99' c
7 (6): < | array pos. now 1
8 (7): . | output '98' b
9 (8): < | array pos. now 0
10 (9): . | output '97' a
```

Zusätzlich zu diesen grundlegenden sechs Befehlen, gibt es noch zwei weitere Befehle, die man benutzen kann, um Schleifen zu bilden. Diese lauten wie folgt

`[` Falls der Inhalt des aktuellen Felds gleich Null ist, überspringe den Code, der zwischen den entsprechenden Klammerpaaren `[ ... ]` steht.

`]` Falls der Inhalt des aktuellen Felds ungleich Null ist, springe zurück zum ersten Befehl, der zwischen zwischen den entsprechenden Klammerpaaren `[ ... ]` steht.

Beispiel 3. Das folgende Programm liest ein Zeichen ein und gibt dann dieses Zeichen, sowie die zwei nachfolgenden Zeichen aus.

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

Gibt man z.B. ‚a' ein, so erhält man als Output ‚abc'. Der Online Brainfuck Interpreter unter `http://koti.mbnet.fi/villes/php/bf.php` ergibt folgenden Debug-Output:

```
0 (0): (The program contains 12 instructions.)

1 (0): , | read in a (97)
2 (1): > | array pos. now 1
3 (2): + | a[1]= 1
4 (3): + | a[1]= 2
5 (4): + | a[1]= 3
6 (5): [ | Array[1] is '3' ** Loop nesting level: 0.
7 (6): < | array pos. now 0
8 (7): . | output '97' a
9 (8): + | a[0]= 98
10 (9): > | array pos. now 1
11 (10): - | a[1]= 2
12 (11): ] | Array[1] is '2'
12 (11): ] | looping back to 5
13 (5): [ | Array[1] is '2' ** Loop nesting level: 0.
14 (6): < | array pos. now 0
15 (7): . | output '98' b
16 (8): + | a[0]= 99
17 (9): > | array pos. now 1
18 (10): - | a[1]= 1
19 (11): ] | Array[1] is '1'
19 (11): ] | looping back to 5
20 (5): [ | Array[1] is '1' ** Loop nesting level: 0.
21 (6): < | array pos. now 0
22 (7): . | output '99' c
23 (8): + | a[0]= 100
24 (9): > | array pos. now 1
25 (10): - | a[1]= 0
26 (11): ] | Array[1] is '0'
```

Beispiel 4. Das folgende Programm wurde mit dem „Text Generator" von „Brainfuck Developer 1.4.7" erzeugt und stellt ein klassisches „Hello, world!"-Programm dar.

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

Es wird empfohlen, dies mit einem Debugger schrittweise auszuführen.

2 Aufgaben

Aufgabe 1. Stellen Sie den Inhalt des Bandes für jeden Schritt der Ausführung des Programms

```
+>++>+++>
```

in der folgenden (oder einer ähnlichen) Form dar

Feld 1	Feld 2	Feld 3	Feld 4	...

wobei die Position des Schreib- und Lesekopfs mittels Unterstreichen markiert wird.

Aufgabe 2. Stellen Sie den Inhalt des Bandes für jeden Schritt der Ausführung des Programms

```
+[+]
```

in der folgenden (oder einer ähnlichen) Form dar

Feld 1	Feld 2	Feld 3	Feld 4	...

wobei die Position des Schreib- und Lesekopfs mittels Unterstreichen markiert wird.

Aufgabe 3. Untersuchen Sie das folgende Programm mit verschiedenen Testeingaben und passen Sie es an, so dass aus der Eingabe ‚TEST' die Ausgabe ‚TtEeSsTt' erzeugt wird.

```
,[++++++++++++++++++++++++++++++++.,]
```

Aufgabe 4. Testen Sie das folgende Programm mit mehreren, jeweils vier Zeichen langen Eingaben.

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

Beschreiben Sie das Verhalten des Programms und passen Sie es an, so dass aus der Eingabe ‚abcd' die Ausgabe ‚bdfh' und aus der Eingabe ‚Test' die Ausgabe ‚Ugvx' erzeugt wird.

Aufgabe 5. Überlegen Sie sich, dass man Addition wie folgt definieren könnte, wenn man nur die Operation +1 zur Verfügung hat:

$$a + b = a \underbrace{+1 + 1 \cdots + 1}_{b\text{-mal}},$$

d.h. zum Beispiel

$$3 + 5 = 3 + 1 + 1 + 1 + 1 + 1.$$

Versuchen Sie diese Art der Addition mit Brainfuck zu implementieren. Ergänzen Sie das folgende Programm, so dass am Schluss die Summe der ersten beiden Felder ins erste Feld geschrieben wird.

```
+++>
+++++
```

Aufgabe 6. Schreiben Sie ein Programm, dass den Wert des ersten Feldes in das zweite Feld kopiert. D.h. wenn man mit

a	0	0	0	...

startet, soll am Schluss

a	a	0	0	...

auf dem Band stehen.

Aufgabe 7. Benutzen Sie die vorherigen beiden Aufgaben, um eine Multiplikation von zwei Zahlen zu implementieren.

Aufgabe 8. Das folgende Programm wurde mit dem „Text Generator" von „Brainfuck Developer 1.4.7" erzeugt und stellt ein klassisches „Hello, world!"-Programm dar.

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

Führen Sie das Programm zuerst schrittweise aus, um dieses zu verstehen. Passen Sie anschliessend das Programm an (ohne den „Text Generator“, so dass es Sie begrüsst, also z.B. „Hello, Samuel!“ ausgibt.

Aufgabe 9. Schreiben Sie ein Programm, dass einen beliebigen String einliest und diesen dann in umgekehrter Reihenfolge ausgibt, d.h. z.B. Input ‚Test‘ ergibt Output ‚tseT‘.

Aufgabe 10. Was ist der Unterschied zwischen

```
+[>+]
```

und

```
+[+>]
```

Aufgabe 11. Erweitern Sie

```
class MyBrainfuckProgram {
    static final int FIELD_SIZE = 1000;

    public static void main(String[] args) {
        int[] field = new int[FIELD_SIZE];
        int currentPosition = 0;

        /* Simulate Brainfuck program here */

    }
}
```

so, dass das folgende Brainfuck-Programm simuliert wird

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