Brainfuck

1 Einleitung

1.1 Vorwort für Lehrpersonen

Das vorliegende Dokument entstand im Rahmen der Lehrplan-orientierten Fachstudien (LOFS) der Fachdidaktik Informatik am Institut Sekundarstufe II der PHBern. Es soll eine kurze Einführung (im Umfang einer Doppellektion) in die Programmiersprache „Brainfuck“ (siehe unten) geben und ist so gestaltet, dass die Unterlagen ohne weitere Bearbeitung übernommen werden können.

Samuel Bucheli
April 2008

1.2 Didaktische Überlegungen

1.2.1 Das Zielpublikum

Das Zielpublikum der folgenden Lektion ist eine fortgeschrittene Klasse im gymnasialen Informatikunterricht, die über gute Programmierkenntnisse verfügt und auch ein gewisses mathematisches Interesse und Lust am Knobeln und Tüfteln mitbringt. Daher sei folgende Warnung vorangestellt: Diese Lektion ist längst nicht für jede Klasse geeignet.

Sollte sich die Lehrperson in der glücklichen Lage finden, eine geeignete Klasse zur Verfügung zu haben, so ist des weiteren darauf zu achten, dass diese Lektion entsprechend in den Informatikunterricht eingebettet wird. Dies bedeutet: Diese Lektion sollte z.B. als Vorbereitung auf Lektionen zum Thema theoretische Informatik (Turing-Maschinen) oder zum Thema Compilerbau verwendet werden. Eine Nutzung in Bezug auf den Bereich technische Informatik (Assemblerprogrammierung) wäre ebenfalls denkbar.

1.2.2 Leitziele

Obwohl sich Computer und damit verbundene Technologien rasant entwickeln und in der Informatikwelt somit nichts langen Bestand zu haben scheint, gibt es doch bei genauerem Hinsehen Konzepte, die für die Ewigkeit (oder zumindest für die nächsten paar hundert Jahre) geschaffen sind. Dazu gehören insbesondere die mathematischen Berechnungsmodelle der theoretischen Informatik und natürlich die Turing-Maschine. Diese wurden teilweise

sogar vor dem Aufkommen von eigentlichen Computern konzipiert und trotzdem konnte bisher niemand ein Modell finden, das aus berechnungstheoretischer Sicht stärker wäre, d.h. heutige Computer können keine anderen Funktionen (im Sinne von Ein- und Ausgabe-Verhalten für Bitmuster) berechnen, als dies Turing-Maschinen können (eigentlich können Turing-Maschinen sogar mehr als heutige Computer, da Turing-Maschinen einen potenziell unendlichen Speicher haben).

Neben der Tatsache, dass uns die Turing-Maschinen eine absolute obere Schranke der Berechenbarkeit geben und somit das prinzipiell Mögliche der Informatik (mit einer Genauigkeit, die wohl kaum in einer anderen Wissenschaft — die Mathematik ausgenommen — möglich ist) aufzeigen, bestechen sie auch durch einen gewissen eleganten Minimalismus und Schlichtheit, die in der Zeit der objektorientierten Programmiersprachen mit gigantischen Klassenbibliotheken und zusätzlichen Frameworks fast wohltuend erscheint.

Des weiteren erweisen sich minimale Programmiersprachen als ideales Laborobjekt in diesem Bereich, ganz im Sinne von „reduced to the max“, die tiefe Einsichten in die Bereiche der theoretischen Informatik (Turing-Berechenbarkeit), der technischen Informatik (einfaches Computermodell auf Hardware-Ebene, Assembler) und der Programmiersprachen (Compilerbau, Interpreter) liefern.

1.2.3 Fundamentale Ideen

Man könnte die fundamentale Idee, welche den Schülerinnen und Schülern vermittelt werden soll, wie folgt zusammenfassen: Trotz der rasanten Entwicklung in der Informatikwelt und trotz aller Unterschiede von Windows über Linux bis Mac OS X und von Visual Basic über C++ bis Java gibt es einen sehr tiefen und allgemeinen Zusammenhang all dieser verschiedenen Plattformen und Programmiersprachen hinsichtlich ihrer Möglichkeiten und Beschränkungen. Insbesondere kann man mit sehr wenigen Mitteln viel komplexere Gegenstände zusammenbauen, so dass diese komplexeren Gegenstände eigentlich redundant sind.

1.2.4 Dispositionsziele

Die Schülerinnen und Schüler sollen ein Gefühl für die Funktionsweise und Programmierung von einfachen Computermodellen bekommen, insbesondere sollen sie dabei Grundlegende Konzepte erkennen, die alle Programmiersprachen und Computer miteinander teilen. Diese Kenntnisse sollen den Einstieg in andere Modelle wie Turing-Maschinen oder Assemblerprogrammierung erleichtern.

1.2.5 Operationalisierte Ziele

- Die Schülerinnen und Schüler kennen Syntax und Semantik von Brainfuck, dies bedingt insbesondere, dass die Schülerinnen und Schüler das Maschinenmodell hinter Brainfuck verstehen.
- Die Schülerinnen und Schüler können einfache Programme, wie z.B. Addition oder Multiplikation analysieren oder selber schreiben.
- Die Schülerinnen und Schüler haben eine Vorstellung davon, wie ein Programm Brainfuck-Quelltext automatisch in Java-Quelltext übersetzen könnte.

1.3 Gestaltung des Unterrichts

Bei der Programmierung mit Brainfuck stellt sich sehr schnell das Problem, dass die einfachen Beispiele schnell ermüdend wirken können, da sich hier vieles wiederholt. Andererseits sind kompliziertere Beispiele oft äusserst umfangreich. Deshalb wird empfohlen, Brainfuck höchstens eine Doppelstunde lang einzusetzen, es könnte sonst schnell langweilig wirken. Einzelne, fortgeschrittenere Schüler sollen dabei aber natürlich ermutigt werden, eigenständig an den komplizierteren Probleme zu tüfteln.

Generell gilt im Informatikunterricht, dass man nicht zu lange über ein Produkt sprechen soll, bevor die Schülerinnen und Schüler dieses ausprobieren können. Daher wird folgender genereller Aufbau empfohlen

1. Kurze (!) Einführung durch Lehrervortrag. Hier soll insbesondere das Maschinenmodell hinter Brainfuck vorgestellt werden. Es empfiehlt sich, z.B. an der Wandtafel ein Modell zu zeichnen (oder sogar mit Holz o.ä. zu basteln) und dann den Schülerinnen und Schülern dieses vorzuführen. Zusätzlich ist natürlich noch ein Hinweis auf die verwendete Entwicklungsumgebung nötig.

2. Schülerinnen und Schülerarbeiten mit Aufgabenblatt (Einzelarbeit oder Partnerarbeit) an PCs, individuelle Beratung und Hilfe durch Lehrperson. Durch die Anzahl und Schwierigkeitsgrade der Aufgaben erfolgt eine Differenzierung. Die Motivation der Schülerinnen und Schüler soll durch Erfolgserlebnisse im Programmieren erfolgen.

3. Zum Schluss der Lektion erfolgt die Konsolidierung des Gelernten im Plenumsgespräch, Schülerinnen und Schüler berichten von ihren Erfahrungen und Erlebnissen mit Brainfuck, Lehrperson schlägt Brücke zu anknüpfendem Unterricht (z.B. Turing-Maschinen) und kann eventuell noch einmal das Umfeld und den Nutzen solcher Modelle noch einmal verdeutlichen, u.a. durch interessante Resultate aus der theoretischen Informatik wie z.B. das Halteproblem oder das P-NP-Problem.

Der kurze Lehrervortrag soll dabei eine ikonische und (falls ein Modell verwendet wird) auch eine enaktive Darstellung bewirken. Durch das Arbeiten mit den Entwicklungsumgebungen und den Programmcodes wird zusätzlich eine virtuell-enaktive und symbolische Darstellung ermöglicht.

Eine Möglichkeit, das Maschinenmodell enaktiv darzustellen, liegt z.B. darin, eine Art „Setzkasten“ zu verwenden (oder ein anderes in mehrere Felder unterteiltes Objekt) und den Inhalt der jeweiligen Felder z.B. mit Murmeln darzustellen. So können die Schülerinnen und Schüler selber den Schreib- und Lesekopf spielen und in die jeweiligen Felder Murmeln hinzufügen oder herausnehmen.

Der Lehrperson obliegt weiter die Verantwortung aus der Aufgabensammlung geeignete Aufgaben auszuwählen. Diese haben verschiedene Schwierigkeitsgrade und fordern auch verschiedene kognitive Niveaus. Die beiden letzten Aufgaben sind äusserst schwierig und sollten daher nur dann benutzt werden, wenn man einzelnen Schülerinnen und Schülern diese ausserordentliche Leistung zutraut. Für diese Schülerinnen und Schüler wird der Gewinn an Erkenntniss dafür umso grösser sein.

1.4 Entwicklungsumgebung für Brainfuck

Wie bei jeder Programmiersprache stellt sich natürlich auch bei Brainfuck die Frage nach der zu verwendenden Entwicklungsumgebung. Eine Übersicht über alle möglichen und unmöglichen Implementationen findet man unter `http://esoteric.voxelperfect.net/wiki/Brainfuck`. Im folgenden finden Sie zwei Vorschläge:

Die IDE „Brainfuck Developer“ ist leider nur für Windows verfügbar, ist allerdings sehr schön gestaltet und kennt auch einen recht komfortablen Debug-Modus, man findet sie unter `http://4mhz.de/bfdev.html`. Bei diesem ist zusätzlich darauf zu achten, dass Eingaben von einzelnen Zeichen zusätzlich mit der Zahl 0 abzuschliessen sind, damit die hier vorgestellten Beispiele funktionieren. Eine Alternative, die überall läuffähig sein sollte ist der Online Brainfuck Interpreter unter `http://koti.mbnet.fi/villes/php/bf.php`. Alle hier vorgestelletn Beispiele und Lösungen wurden mit diesem Interpreter getestet und sollten funktionieren.

Des weiteren möchte ich hier ebenfalls darauf hinweisen, dass sich verschiedene Implementationen von Brainfuck recht unterschiedlich verhalten können. Z.B. wird in einigen der unten stehenden Programme ausgenutzt, dass ein Schritt vom ersten Feld nach links ins letzte Feld führt (dass dann den Wert Null enthalten sollte). Bei einigen Brainfuck Interpretern wird dies aber zu einer Fehlermeldung führen. Damit diese Programme funktioneren kann man z.B. einfach jeweils am Anfang noch ein `>` einfügen. Es gibt noch einige andere Spezialitäten dieser Art, aber auf diese einzugehen würde die ganze Sache nur unnötig kompliziert machen. Mit ein paar einfachen Testprogrammen kann man schnell herausfinden, wie sich eine Brainfuckimplementation verhält (es empfiehlt sich sogar, dies den Schülerinnen und Schülerim Sinne von entdeckendem Lernen als Aufgabe zu überlassen).

1.5 Quellen

Hier sollen noch einmal alle Quellen zu Brainfuck zusammengestellt werden, die ich als nützlich ansehe:

- `http://de.wikipedia.org/wiki/Brainfuck`
- `http://de.wikipedia.org/wiki/Esoterische_Programmiersprache`
- `http://esoteric.voxelperfect.net/wiki/Brainfuck`
- `http://esoteric.sange.fi/brainfuck/`
- `http://4mhz.de/bfdev.html`
- `http://koti.mbnet.fi/villes/php/bf.php`

2 Brainfuck

2.1 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.

2.2 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.

2.3 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.

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

- 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.

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 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
```

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.

3 Aufgaben und Lösungen

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.

Lösung 1. Eine mögliche Lösung:

0	0	0	0	...
1	0	0	0	...
1	0	0	0	...
1	1	0	0	...
1	2	0	0	...
1	2	0	0	...
1	2	1	0	...
1	2	2	0	...
1	2	3	0	...
1	2	3	0	...

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

Die entsprechenden Befehle wurden zusätzlich in der rechten Seite vermerkt.

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.

Lösung 2. Eine mögliche Lösung:

0	0	0	0	...
1	0	0	0	...
2	0	0	0	...
3	0	0	0	...
4	0	0	0	...
5	0	0	0	...

```
+
[+]
[+]
[+]
[+]
```

...

d.h. das Programm gerät in eine Endlosschleife.

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.

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

Lösung 3. Eine mögliche Lösung:

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

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.

Lösung 4. Das Programm implementiert eine sogenannte Caesar-Shift-Verschlüsselung, bei der jeder Buchstabe um eine Stelle „nach rechts" verschoben wird.

Für den zweiten Teil der Aufgabe muss das erste Zeichen um eine Stelle, das zweite Zeichen um zwei Stellen, das dritte um drei Stellen und das vierte Zeichen um vier Stellen verschoben werden. Dies kann z.B. wie folgt gelöst werden:

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

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.

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

Lösung 5. Eine mögliche Lösung:

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

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.

Lösung 6. Der Trick besteht darin, dass man zuerst den Wert des ersten Feldes in die folgenden zwei Felder (ähnlich wie bei der Addition) verschiebt, also aus

a	0	0	0	...

wird

0	a	a	0	...

und dann den Wert des dritten Feldes ins erste zurückkopiert, also

a	a	0	0	...

Dies kann man z.B. wie folgt lösen:

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

(In der ersten Zeile wird der zu kopierende Wert eingeben, in diesem Fall also 4)

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

Lösung 7. Ähnlich wie bei der Addition kann man die Multiplikation wie folgt auffassen

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

also z.B.

$$3 \cdot 5 = 5 + 5 + 5.$$

Wir benötigen also eine zweite Schleife, innerhalb dieser wird die Addition dann genügend oft ausgeführt. Damit wir immer die richtigen Additionswerte zur Verfügung haben, muss dieser zuerst in ein entsprechendes Feld kopiert werden.

Eine mögliche Lösung[2] sieht wie folgt aus:

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

[2] von `http://de.wikipedia.org/wiki/Brainfuck`

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.

Lösung 8. Eine mögliche Lösung:

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

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‘.

Lösung 9. Eine mögliche Lösung:

```
,[>,]<[.<]
```

Aufgabe 10. Was ist der Unterschied zwischen

```
+[>+]
```

und

```
+[+>]
```

Lösung 10. Das erste Programm gerät in eine Endlosschleife, es füllt jedes Band auf dem Feld mit einer Eins, also

0	0	0	0	...
1	0	0	0	...
1	1	0	0	...
1	1	1	0	...
1	1	1	1	...

...

Das zweite Programm durchläuft die Schleife genau einmal und bleibt im ersten Feld, also

0	0	0	0	...
1	0	0	0	...
2	0	0	0	...

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

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

Lösung 11. Eine mögliche Lösung:

```
1 class MyBrainfuckProgram {
2     static final int FIELD_SIZE = 1000;
3
4     public static void main(String[] args) {
5         int[] field = new int[FIELD_SIZE];
6         int currentPosition = 0;
```

```
7
8           field[currentPosition]++;
9           field[currentPosition]++;
10          field[currentPosition]++;
11          currentPosition++;
12          field[currentPosition]++;
13          field[currentPosition]++;
14          field[currentPosition]++;
15          field[currentPosition]++;
16          field[currentPosition]++;
17
18          while(field[currentPosition]!=0) {
19              currentPosition--;
20              field[currentPosition]++;
21              currentPosition++;
22              field[currentPosition]--;
23          }
24
25      }
26  }
```

Die folgende Lösung läuft online unter `http://clab2.phbern.ch:81/lego/Communication.html`

```
1   /* Runs in http://clab2.phbern.ch:81/lego/Communication.html */
2
3   import ch.aplu.util.*;
4
5   class MyBrainfuckProgramApluVersion extends Console {
6       static final int FIELD_SIZE = 1000;
7
8       public static void main(String[] args) {
9           int[] field = new int[FIELD_SIZE];
10          int currentPosition = 0;
11
12          field[currentPosition]++;
13          field[currentPosition]++;
14          field[currentPosition]++;
15          currentPosition++;
16          field[currentPosition]++;
17          field[currentPosition]++;
18          field[currentPosition]++;
19          field[currentPosition]++;
20          field[currentPosition]++;
21
22          while(field[currentPosition]!=0) {
23              currentPosition--;
24              field[currentPosition]++;
25              currentPosition++;
```

```
26                  field[currentPosition]--;
27              }
28
29              print("Der Wert im ersten Feld ist "+field[0]);
30          }
31  }
```

4 Anhang: Zwei sehr schwere Aufgaben

Aufgabe 12. Schreiben Sie einen Interpreter oder Compiler für Brainfuck.

Lösung 12. Die folgenden beiden Beispiele stammen von `http://esoteric.sange.fi/brainfuck/`. Das erste ist ein Interpreter für Brainfuck, das zweite ist ein Compiler für Brainfuck, der Brainfuck-Programme in Java-Programme übersetzt. Beide Beispiele sind in Java geschrieben. Unter der oben genannten Adresse finden sich viele weitere Implementationen von Compilern und Interpretern in verschiedensten Sprachen.

Der Interpeter:

```
1  /*
2   *  BFI
3   *  Copyright (C) 2003 Thomas Cort
4   *
5   *  This program is free software; you can redistribute it and/or
        modify
6   *  it under the terms of the GNU General Public License as published
         by
7   *  the Free Software Foundation; either version 2 of the License, or
8   *  (at your option) any later version.
9   *
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
        02111-1307 USA
18  */
19
20 import java.io.BufferedReader;
21 import java.io.FileReader;
22 import java.io.InputStreamReader;
23
24 /**
25  *  BFI - Compliant with The ENSI BrainFuck Language Specification v1
        .3, and
26  *  also compliant with The Portable Brainfuck Specification.
27  *  @author Thomas Cort (<A HREF="mailto:tom@tomcort.com">tom@tomcort
        .com</A>)
28  *  @version 1.1 2003-03-16
29  *  @since   1.0
30  */
31 public class BFI {
32   protected final int MAX_DATA_SIZE = 32767;
33   protected byte [] x;
```

```
34    protected char [] c;
35    protected int p, pc, l;
36
37    /**
38     *  Constructor
39     *  @param String - Brainfuck program to interpret.
40     *  @since 1.0
41     */
42    public BFI (String s) {
43      x = new byte[MAX_DATA_SIZE+1];
44      c = s.toCharArray();
45      for (int i = 0; i < x.length; i++) x[i] = 0;
46      p = l = pc = 0;
47    }
48
49    /**
50     *  Evaluates the current character in the Brainfuck program
51     *  @since 1.0
52     */
53    public void interpret() {
54      for (pc = 0; pc < c.length ; pc++) {
55        if      (c[pc] == '>') incrementPointer();
56        else if (c[pc] == '<') decrementPointer();
57        else if (c[pc] == '+') incrementByteAtPointer();
58        else if (c[pc] == '-') decrementByteAtPointer();
59        else if (c[pc] == '.') output();
60        else if (c[pc] == ',') input();
61        else if (c[pc] == '[') startJump();
62        else if (c[pc] == ']') endJump();
63      }
64    }
65
66    /**
67     *  Checks sytanx. Must be called explicitly before you call
          interpret() if
68     *  you want to check the syntax.
69     *  @since 1.0
70     */
71    public void checkSyntax() {
72      int lt = 0, gt = 0, ob = 0, cb = 0, ptrAt = 0;
73
74      for (int t = 0; t < c.length; t++) {
75          if      (c[t] == '[') ob++;
76          else if (c[t] == ']') cb++;
77      }
78
79      if (ob != cb)
80        errBF( ob>cb ? "Missing␣closing␣bracket(s)":"Missing␣opening␣
```

```
            bracket(s)");
81    }
82
83    /**
84     *  '>' Increment pointer.
85     *  @since 1.0
86     */
87    public void incrementPointer() {
88      if (++p > MAX_DATA_SIZE)
89        errBF("Pointer moved beyond MAX_DATA_SIZE: " + MAX_DATA_SIZE);
90    }
91
92    /**
93     *  '<' Decrement pointer.
94     *  @since 1.0
95     */
96    public void decrementPointer() {
97            if (--p < 0)
98        errBF("Pointer moved to the left of the starting position");
99    }
100
101   /**
102    *  '+' Increments the value at the pointer.
103    *  @since 1.0
104    */
105   public void incrementByteAtPointer() {
106     if (x[p] == 127)
107       errBF("Byte incremented beyond capacity");
108     ++x[p];
109   }
110
111   /**
112    *  '-' Dencrements the value at the pointer.
113    *  @since 1.0
114    */
115   public void decrementByteAtPointer() {
116     if (x[p] == -128)
117       errBF("Byte decremented below capacity");
118     --x[p];
119   }
120
121   /**
122    *  '[' Start of loop code block.
123    *  @since 1.0
124    */
125   public void startJump() {
126         if (x[p] == 0) {
127           pc++;
```

```
128             while (l > 0 || c[pc] != ']') {
129           if (c[pc] == '[') l++;
130           if (c[pc] == ']') l--;
131           pc++;
132             }
133       }
134     }
135
136     /**
137      *  ']' End of loop code block.
138      *  @since 1.0
139      */
140     public void endJump() {
141       pc--;
142       while (l > 0 || c[pc] != '[') {
143         if (c[pc] == ']') l++;
144         if (c[pc] == '[') l--;
145         pc--;
146       }
147       pc--;
148     }
149
150     /**
151      *  '.' converts current byte to ascii & prints it to STDOUT
152      *  @since 1.0
153      */
154     public void output() {
155           byte [] out = {x[p]};
156           String s = new String(out);
157           System.out.print(s);
158     }
159
160     /**
161      *  ',' reads a character from STDIN, and stores it at the current
         location
162      *  @since 1.0
163      */
164     public void input() {
165             BufferedReader Stream =
166               new BufferedReader(new InputStreamReader(System.in));
167
168             try {
169         String str = Stream.readLine();
170         byte [] in = str.getBytes();
171         x[p] = in[0];
172             } catch (Exception e) {
173               errBF("Input␣Parse␣Error");
174             }
```

```
175     }
176
177     /**
178      *  Error Handler
179      *  @param String - error message
180      *  @since 1.0
181      */
182     public void errBF(String str) {
183       throw new Error(str);
184     }
185
186
187     public static void main(String [] args) throws Exception {
188       BFI b = new BFI("");
189       BufferedReader reader;
190       String line = "", input = "", filename = "";
191
192       // Read each file and interpret.
193       for(int z = 0; z < args.length; z++) {
194         filename = args[z];
195         input = "";
196
197         try {
198           reader = new BufferedReader( new FileReader( filename ) );
199           while ( (line = reader.readLine()) != null )
200             input += line;
201         } catch (Exception e) {
202           b.errBF("Cannot read input file");
203         }
204
205         b = new BFI(input);
206         b.checkSyntax();
207         b.interpret();
208       }
209     }
210 }
```

Der Compiler:

```
1  /*
2   *  BF2Java
3   *  Copyright (C) 2003 Thomas Cort
4   *
5   *  This program is free software; you can redistribute it and/or
        modify
6   *  it under the terms of the GNU General Public License as published
        by
7   *  the Free Software Foundation; either version 2 of the License, or
8   *  (at your option) any later version.
9   *
```

```
10  *  This program is distributed in the hope that it will be useful,
11  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
12  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13  *  GNU General Public License for more details.
14  *
15  *  You should have received a copy of the GNU General Public License
16  *  along with this program; if not, write to the Free Software
17  *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
        02111-1307 USA
18  */
19
20 /*
21  * Program Name:   BF2Java
22  * Version:        1.0
23  * Date:           2003-03-18
24  * Description:    Converts Brainfuck source to Java source
25  * License:        GPL
26  * Web page:       http://www.brainfuck.ca
27  * Download:       http://www.brainfuck.ca/BF2Java.java
28  * Source Info:    http://www.brainfuck.ca/downloads.html
29  * Latest Ver:     http://www.brainfuck.ca/downloads.html
30  * Documentation: None
31  * Help:           tom@brainfuck.ca
32  * Developement:   tom@brainfuck.ca
33  * Bugs:           tom@brainfuck.ca
34  * Maintainer:     Thomas Cort <tom@brainfuck.ca>
35  * Developer:      Thomas Cort <tom@brainfuck.ca>
36  * Interfaces:     Command Line
37  * Source Lang:    Java
38  * Build Prereq:   None
39  * Related Progs: BF2C
40  * Category:       Software Development > Programming language
        conversion
41  */
42
43 import java.io.BufferedReader;
44 import java.io.FileReader;
45
46 /**
47  *  BF2Java - convert Brainfuck to Java
48  *  @author Thomas Cort (<A HREF="mailto:tom@tomcort.com">tom@tomcort
        .com</A>)
49  *  @version 1.1 2003-03-16
50  *  @since   1.0
51  */
52 public class BF2Java {
53
54   /**
```

```
55      *  Converts Brainfuck to Java
56      *  @param  String - BF  code
57      *  @return String - Java code
58      *  @since 1.0
59      */
60     public static String convert(String s) {
61
62 String javaCode = "" +
63 "import java.io.BufferedReader;\n" +
64 "import java.io.InputStreamReader;\n" +
65 "\n" +
66 "public class output {\n" +
67 "\n" +
68 "  public static void putchar(byte c) {\n" +
69 "    byte [] out = {c};\n" +
70 "    String s = new String(out);" +
71 "    System.out.print(s);" +
72 "  }" +
73 "\n" +
74 "public static byte getchar() {\n" +
75 "  BufferedReader Stream = \n" +
76 "    new BufferedReader(new InputStreamReader(System.in));\n" +
77 "\n" +
78 "  try {\n" +
79 "    String str = Stream.readLine();\n" +
80 "    byte [] in = str.getBytes();\n" +
81 "    return in[0];\n" +
82 "  } catch (Exception e) {\n" +
83 "    throw new Error(\"Input Parse Error\");\n" +
84 "  }\n" +
85 "}\n" +
86 "\n" +
87 "public static void main(String [] args) {\n"+
88 "  int pc = 0;\n" +
89 "  byte [] x = new byte[32768];\n";
90
91      for(int i = 0; i < s.length(); i++) {
92        if      (s.charAt(i) == '>') javaCode += "    pc++;\n";
93        else if (s.charAt(i) == '<') javaCode += "    pc--;\n";
94        else if (s.charAt(i) == '+') javaCode += "    x[pc]++;\n";
95        else if (s.charAt(i) == '-') javaCode += "    x[pc]--;\n";
96        else if (s.charAt(i) == '.') javaCode += "    putchar(x[pc]);\n
              ";
97        else if (s.charAt(i) == ',') javaCode += "    x[pc] = getchar()
              ;\n";
98        else if (s.charAt(i) == '[') javaCode += "    while (x[pc] != 
              0) {\n";
99        else if (s.charAt(i) == ']') javaCode += "    }\n";
```

```
100       }
101
102       return javaCode + "}\n}";
103     }
104
105     /**
106      *  Error Handling
107      *  @param String - error message
108      *  @since 1.0
109      */
110     public static void errBF2Java(String s) {
111       throw new Error(s);
112     }
113
114     public static void main(String [] args) throws Exception {
115
116       BufferedReader reader;
117       String line = "", input = "", filename = "";
118
119       // Read each file and interpret.
120       for(int z = 0; z < args.length; z++) {
121         filename = args[z];
122         input = "";
123
124         try {
125           reader = new BufferedReader( new FileReader( filename ) );
126           while ( (line = reader.readLine()) != null )
127             input += line;
128         } catch (Exception e) {
129           errBF2Java("Cannot read input file");
130         }
131
132         System.out.println(BF2Java.convert(input));
133       }
134     }
135
136 }
```

Aufgabe 13. Schreiben Sie ein Brainfuck-Programm, das seinen eigenen Quelltext ausgibt.

Lösung 13. Ein solches Programm nennt man übrigens Quine, siehe `http://de.wikipedia.org/wiki/Quine_(Computerprogramm)`. Im folgenden sehen Sie eine Lösung von `http://esoteric.sange.fi/brainfuck/`. Anschliessend an diese folgt eine kommentierte Version dieser Lösung (dieses Programm ist natürlich kein Quine mehr, es müsste ja auch die Kommentare ausgeben). Um eine bessere Lesbarkeit zu ermöglichen, wurden Zeilen automatisch umgebrochen. Das originale Quine hat keine Zeilenumbrüche, sondern ist eine sehr lange Zeile.

Das Quine (mit zusätzlichen Zeilenumbrüchen):

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

Das kommentierte Quine (das eigentlich kein Quine mehr ist):

```
1  Written by Erik Bosman
2
3  a 410 byte quine with comments
4
5  DATA SECTION
6
7  : The data section contains the data needed for reproducing
       the code section
8  : The format in which the data is stored is as follows:
9  :
10 : every character is stored into memory as two separate bytes
       (B1; B2)
11 : the character that is represented by these bytes has the
       ascii value
12 : B1 plus B2*16 plus 43
13 :
14 : I chose this format because it makes it possible to write a
       very compact
15 : code section which has its effect on the size of the data
       section too
16 :
17 : The data is inserted in reversed order and the first minus
       sign causes the
18 : code to stop at the beginning of the stream
19
20 -
21 >++>+++ >+>+ >++> >+>+ >+++> >+>+ >++>+++ >+++>+ >> >> >> >>
       >> >> >> >> >> >>
22 >> >> >> >> >> >> >+>+ >++> >>+++ >> >> >+++>+ >> >> >> >> >>
       >> >> >> >> >>
23 >>+++ >> >> >> >++>+++ >+++>+ >>+++ >> >+++>+ >+++>+ >++>+++
       >> >+>+ >+>+
```

```
24 >++>+++ >+>+ >>+++ >> >> >> >+>+ >> >+>+ >++>+++ >+++>+ >>+++
       >+++>+ >+++>+
25 >++>+++ >++> >+>+ >++>+++ >+>+ >>+++ >> >+++>+ >> >++>+++
       >+++>+ >>+++ >>
26 >+++>+ >+++>+ >>+++ >>+++ >>
27
28
29 CODE SECTION
30
31 : The data section contains information about how to print the
       code section
32 : The following code examines this data and uses it to add the
       code needed
33 : for the insertion of the data into the memory (the data
       section)
34 : ( this data is in the same format as the code section and
       also in
35 :   reversed order )
36 :
37 : In the process all bytes are increased by one
38 : therefore the ascii value of a character can now be
       calculated by
39 : B1 plus B2*16 plus 26
40
41 WHILE THERE ARE BYTES LEFT: +[
42         : move the data two positions to the right
43         : and add N plus signs at the end where N
44         : is the value of the current byte
45
46         [
47                 >>+[>]+>+[<]<-
48         ]
49
50         : note that this actually is one too many because the
              byte
51         : already has been increased once
52         :
53         : go to the end of the stream once again to replace
              the last plus sign
54         : by a greater than sign and return to the orginal
              position
55
56         >>[>]<+<+++[<]<
57
58         : on to the next byte
59         <
```

```
+]


: Go to the end and add the data to print a minus sign there
:
: The third character of the code below is a plus sign instead of a
: greater than sign because it uses less space in the data section
: and it works just as well; the value of this byte is irrelevant
: in the code that is to follow

>>+[>]+++


: note that B1 is 3 and B2 is 0 and the ascii character of the minus sign
: is 45 and not 29 as the formula as described above would return
: This is because in the following loop B2 will be decreased by one for
: all characters but the first one
:
: The actual formula applied will therefore be:
: (B1 plus 10) plus (B2 plus 2)*16

WHILE THERE IS A CHARACTER LEFT TO PRINT: [
        ++++++++++>++
        [-<++++++++++++++++>]
        <.<-<
]
```

Es ist übrigens beweisbar, dass Quines für jede Turing-vollständige Programmiersprache existieren, es handelt sich hierbei nicht um einen „Design-Unfall“ von Brainfuck. Das ganze beruht im wesentlich auf Kleenes zweitem Rekursionstheorem, weitere Informationen dazu findet man unter `http://de.wikipedia.org/wiki/Quine_(Computerprogramm)` und `http://de.wikipedia.org/wiki/Rekursionssatz` oder in jedem vernünftigen Text über Berechenbarkeitstheorie.