

Informatik I: Einführung in die Programmierung

20. Finale: Ein Interpreter für Brainf*ck

Albert-Ludwigs-Universität Freiburg



UNI
FREIBURG

Prof. Dr. Peter Thiemann

07.02.2023



Motivation

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Jeder *Informatiker* sollte **mindestens 2 Programmiersprachen** beherrschen!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Jeder *Informatiker* sollte **mindestens 2 Programmiersprachen** beherrschen!
- Python, C++, Scheme, Java, JavaScript, Ruby, Haskell, ...

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Jeder *Informatiker* sollte **mindestens 2 Programmiersprachen** beherrschen!
- Python, C++, Scheme, Java, JavaScript, Ruby, Haskell, ...
- Wir lernen heute eine minimale Programmiersprache kennen, ...

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Jeder *Informatiker* sollte **mindestens 2 Programmiersprachen** beherrschen!
- Python, C++, Scheme, Java, JavaScript, Ruby, Haskell, ...
- Wir lernen heute eine minimale Programmiersprache kennen, ...
- ...bauen dazu einen **Interpreter**,

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Jeder *Informatiker* sollte **mindestens 2 Programmiersprachen** beherrschen!
- Python, C++, Scheme, Java, JavaScript, Ruby, Haskell, ...
- Wir lernen heute eine minimale Programmiersprache kennen, ...
- ...bauen dazu einen **Interpreter**,
- ...der **Dictionaries** und **Exceptions** clever verwendet.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Jeder *Informatiker* sollte **mindestens 2 Programmiersprachen** beherrschen!
- Python, C++, Scheme, Java, JavaScript, Ruby, Haskell, ...
- Wir lernen heute eine minimale Programmiersprache kennen, ...
- ...bauen dazu einen **Interpreter**,
- ...der **Dictionaries** und **Exceptions** clever verwendet.
- ...und wir dürfen uns freuen, dass wir bisher eine sehr viel komfortablere Sprache verwendet haben.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Brainf*ck kennt ganze 8 Befehle

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Brainf*ck kennt ganze 8 Befehle
- Beschrieben 1993 von Urban Müller, der dafür einen Compiler in 240 Byte geschrieben hat.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Brainf*ck kennt ganze 8 Befehle
- Beschrieben 1993 von Urban Müller, der dafür einen Compiler in 240 Byte geschrieben hat.
- Brainf*ck ist **Turing-vollständig**, d.h. alle *berechenbaren Funktionen* können implementiert werden.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Brainf*ck kennt ganze 8 Befehle
- Beschrieben 1993 von Urban Müller, der dafür einen Compiler in 240 Byte geschrieben hat.
- Brainf*ck ist **Turing-vollständig**, d.h. alle *berechenbaren Funktionen* können implementiert werden.
- Eine „esoterische“ Programmiersprache. Andere Vertreter z.B. *Whitespace*, *Chef*, *TrumpScript* und *Shakespeare*.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Programmiersprache

Motivation

Program-
miersprache

Befehle

Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Syntax von Brainf*ck

- **Programm** ist Folge von ASCII-Zeichen (Unicode-Wert 0 bis 127).
- Bedeutungstragend sind aber nur die acht Zeichen:

< > + - . , []
- Alle anderen Zeichen sind Kommentar.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Syntax von Brainf*ck

- **Programm** ist Folge von ASCII-Zeichen (Unicode-Wert 0 bis 127).
- Bedeutungstragend sind aber nur die acht Zeichen:
$$\langle \rangle + - . , []$$
- Alle anderen Zeichen sind Kommentar.

Berechnungsmodell

- Ein Programm wird Zeichen für Zeichen abgearbeitet, bis das Ende des Programms erreicht wird.
- Es gibt einen ASCII-Eingabestrom und einen ASCII-Ausgabestrom (normalerweise die Konsole)
- Die **Daten** werden in Speicherzellen gehalten: data. (Array)
- Es gibt einen **Datenzeiger**, der initial 0 ist: ptr.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Befehle:	$B = [0, 127]$
Befehlszähler:	$pc \in \mathbb{N}$
Datenzeiger/aktuelle Zelle:	$ptr \in \mathbb{N}$
Programm:	$src \in \mathbb{N} \leftrightarrow B$
Datenzellen:	$data \in \mathbb{N} \rightarrow \mathbb{N}$

Der Zustandsraum ist ein Tupel

$$(pc, ptr, src, data, \dots) \in Z \tag{1}$$

mit Startzustand

$$(0, 0, src, \lambda n : 0, \dots) \tag{2}$$

Jeder Befehl beschreibt einen Zustandsübergang $I(B) \in Z \leftrightarrow Z$.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Jeder Befehl wirkt auf $(pc, ptr, src, data, \dots) =$

> Bewege den Datenzeiger nach rechts: $(pc + 1, ptr + 1, src, data, \dots)$

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Jeder Befehl wirkt auf $(pc, ptr, src, data, \dots) =$

- > Bewege den Datenzeiger nach rechts: $(pc + 1, ptr + 1, src, data, \dots)$
- < Bewege den Datenzeiger nach links: $(pc + 1, ptr - 1, src, data, \dots)$

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Jeder Befehl wirkt auf $(pc, ptr, src, data, \dots) =$

- > Bewege den Datenzeiger nach rechts: $(pc + 1, ptr + 1, src, data, \dots)$
- < Bewege den Datenzeiger nach links: $(pc + 1, ptr - 1, src, data, \dots)$
- + Erhöhe den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr \mapsto data[ptr] + 1], \dots)$

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Jeder Befehl wirkt auf $(pc, ptr, src, data, \dots) =$

- > Bewege den Datenzeiger nach rechts: $(pc + 1, ptr + 1, src, data, \dots)$
- < Bewege den Datenzeiger nach links: $(pc + 1, ptr - 1, src, data, \dots)$
- + Erhöhe den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr] \mapsto data[ptr] + 1, \dots)$
- Erniedrige den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr] \mapsto data[ptr] - 1, \dots)$

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Jeder Befehl wirkt auf $(pc, ptr, src, data, \dots) =$

- > Bewege den Datenzeiger nach rechts: $(pc + 1, ptr + 1, src, data, \dots)$
- < Bewege den Datenzeiger nach links: $(pc + 1, ptr - 1, src, data, \dots)$
- + Erhöhe den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr] \mapsto data[ptr] + 1, \dots)$
- Erniedrige den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr] \mapsto data[ptr] - 1, \dots)$
- . Gebe ein ASCII-Zeichen entsprechend dem Wert in der aktuellen Zelle aus:
`print(chr(data[ptr]), end='')`.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Jeder Befehl wirkt auf $(pc, ptr, src, data, \dots) =$

- > Bewege den Datenzeiger nach rechts: $(pc + 1, ptr + 1, src, data, \dots)$
- < Bewege den Datenzeiger nach links: $(pc + 1, ptr - 1, src, data, \dots)$
- + Erhöhe den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr] \mapsto data[ptr] + 1, \dots)$
- Erniedrige den Wert in der aktuellen Zelle:
 $(pc + 1, ptr, src, data[ptr] \mapsto data[ptr] - 1, \dots)$
- . Gebe ein ASCII-Zeichen entsprechend dem Wert in der aktuellen Zelle aus:
`print(chr(data[ptr]), end='')`.
- , Lese ein ASCII-Zeichen und lege den Wert in der aktuellen Zelle ab:
`data[ptr] = sys.stdin.read(1)`.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Ein Programm ohne Verzweigungen und Schleifen, das einen Großbuchstaben in den entsprechenden Kleinbuchstaben übersetzt.

```
konv.b
```

```
Lese ein Zeichen (Annahme: Grossbuchstabe)
```

```
,
```

```
Konvertiere in Kleinbuchstabe
```

```
+++++
```

```
Gebe das Zeichen aus
```

```
.
```

```
Und hier ist das Programm zu Ende
```

Probiere aus auf: <https://brainfuck-visualizer.herokuapp.com/>

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Aus „normalen“ Programmiersprachen kennen wir die `while`-Schleife.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Aus „normalen“ Programmiersprachen kennen wir die `while`-Schleife.
- Diese Rolle spielt in Brainf*ck das Klammerpaar `[und]`:

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Aus „normalen“ Programmiersprachen kennen wir die `while`-Schleife.
- Diese Rolle spielt in Brainf*ck das Klammerpaar `[und]`:
 - [Falls Inhalt der aktuellen Zelle = 0 ist ($data[ptr] = 0$), dann springe vorwärts zum Befehl nach der **zugehörigen schließenden** Klammer (beachte Klammersregeln). Ansonsten setze die Ausführung mit dem Befehl nach der öffenden Klammer fort.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Aus „normalen“ Programmiersprachen kennen wir die `while`-Schleife.
- Diese Rolle spielt in Brainf*ck das Klammerpaar `[` und `]`:
 - [Falls Inhalt der aktuellen Zelle = 0 ist ($data[ptr] = 0$), dann springe vorwärts zum Befehl nach der **zugehörigen schließenden** Klammer (beachte Klammerungsregeln). Ansonsten setze die Ausführung mit dem Befehl nach der öffenden Klammer fort.
 -] Springe zurück zur **zugehörigen öffnenden** Klammer.

Motivation

Program-
miersprache

Befehle
Schleifen

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Beispiele

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



loop.b

```
+++++      set cell #0 to 6
[ > ++++++ add 8 to cell #1
  < -      decrement loop counter cell #0
]
> +        add another 1 to cell #1
.          print ASCII 49 = '1'

-          now cell #1 is '0'
< ++++++  set cell #0 to 8
[ > .      print ASCII 48 = '0'
  < -      decrement loop counter (cell #0)
]
```

Ausgabe: 100000000

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung

Hello World (1)



hello.b - Part 1

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

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung

Hello World (2)



hello.b - Part 2

```
> ++ .          print ' '
<< ++++++ ++++++ ++++++ . print 'W'
> .            print 'o'
+++ .         print 'r'
----- - .   print 'l'
----- ---- . print 'd'
> + .         print '!'
> .          print '\n'
```

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.
 - Auf Null setzen (negative Werte sollten nicht auftreten!): [-]

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.
 - Auf Null setzten (negative Werte sollten nicht auftreten!): [-]
 - Zuweisung von Konstanten an Variable ist einfach: [-]+++ ...
(ggf. Schleife verwenden)

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.
 - Auf Null setzen (negative Werte sollten nicht auftreten!): [-]
 - Zuweisung von Konstanten an Variable ist einfach: [-]+++ ...
(ggf. Schleife verwenden)
 - Addieren des Wertes der aktuellen Zelle zu einer anderen Zelle, (mit gegebenem Abstand, z.B. +3): [->>> + <<<]

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.
 - Auf Null setzen (negative Werte sollten nicht auftreten!): [-]
 - Zuweisung von Konstanten an Variable ist einfach: [-]+++ ...
(ggf. Schleife verwenden)
 - Addieren des Wertes der aktuellen Zelle zu einer anderen Zelle, (mit gegebenem Abstand, z.B. +3): [->>> + <<<]
 - Transfer des Wertes, falls initialer Wert der Zielzelle = 0.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.
 - Auf Null setzen (negative Werte sollten nicht auftreten!): [-]
 - Zuweisung von Konstanten an Variable ist einfach: [-]+++ ...
(ggf. Schleife verwenden)
 - Addieren des Wertes der aktuellen Zelle zu einer anderen Zelle, (mit gegebenem Abstand, z.B. +3): [->>> + <<<]
 - Transfer des Wertes, falls initialer Wert der Zielzelle = 0.
 - Übertragen in zwei Zellen: [->>>+>+<<<<]

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Fast alle Operationen werden durch kleine Programmstücke simuliert.
 - Auf Null setzen (negative Werte sollten nicht auftreten!): [-]
 - Zuweisung von Konstanten an Variable ist einfach: [-]+++ ...
(ggf. Schleife verwenden)
 - Addieren des Wertes der aktuellen Zelle zu einer anderen Zelle, (mit gegebenem Abstand, z.B. +3): [->>> + <<<]
 - Transfer des Wertes, falls initialer Wert der Zielzelle = 0.
 - Übertragen in zwei Zellen: [->>>+>+<<<<<]
 - Kopieren: Erst in zwei Zellen transferieren, dann den einen Wert zurück transferieren.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Kontrollstrukturen und logische Operatoren:
 - *If*-Anweisung ($x \neq 0$):

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Kontrollstrukturen und logische Operatoren:

- *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Kontrollstrukturen und logische Operatoren:

- *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)
 - Annahme, Testvariable ist aktuelle Zelle: [[-] ...]

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

■ *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)

- Annahme, Testvariable ist aktuelle Zelle: [[-] ...]

- Für die logischen Operatoren sei $0 = \textit{False}$, alles andere *True*.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

■ *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)

- Annahme, Testvariable ist aktuelle Zelle: [[-] ...]

■ Für die logischen Operatoren sei $0 = \textit{False}$, alles andere *True*.

■ Logisches Und:

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

■ *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)
- Annahme, Testvariable ist aktuelle Zelle: [[-] ...]

■ Für die logischen Operatoren sei $0 = \textit{False}$, alles andere *True*.

■ Logisches Und:

- Setze Ergebnisvariable auf Null. Dann ein *If*-Statement über dem ersten Operanden, in dem der zweite Operand auf die Ergebnisvariable transferiert wird.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

■ *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)
- Annahme, Testvariable ist aktuelle Zelle: [[-] ...]

■ Für die logischen Operatoren sei $0 = \text{False}$, alles andere *True*.

■ Logisches Und:

- Setze Ergebnisvariable auf Null. Dann ein If-Statement über dem ersten Operanden, in dem der zweite Operand auf die Ergebnisvariable transferiert wird.
- Annahme, Linker Op. aktuell, rechter Op. +1, Ergebnis +2:
>>[-]<< [[-] > [- > + <] <] >>

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

■ *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)
- Annahme, Testvariable ist aktuelle Zelle: `[[-] ...]`

■ Für die logischen Operatoren sei $0 = \textit{False}$, alles andere *True*.

■ Logisches Und:

- Setze Ergebnisvariable auf Null. Dann ein *If*-Statement über dem ersten Operanden, in dem der zweite Operand auf die Ergebnisvariable transferiert wird.
- Annahme, Linker Op. aktuell, rechter Op. +1, Ergebnis +2:
`>>[-]<< [[-] > [- > + <] <] >>`

■ Logisches Oder: Transferiere beide Operanden zur Ergebnisvariable.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Kontrollstrukturen und logische Operatoren:

■ *If*-Anweisung ($x \neq 0$):

- Benutze Schleife und setze die Test-Variable auf Null (ist destruktiv für die getestete Variable!)
- Annahme, Testvariable ist aktuelle Zelle: `[[-] ...]`

■ Für die logischen Operatoren sei $0 = \textit{False}$, alles andere *True*.

■ Logisches Und:

- Setze Ergebnisvariable auf Null. Dann ein *If*-Statement über dem ersten Operanden, in dem der zweite Operand auf die Ergebnisvariable transferiert wird.
- Annahme, Linker Op. aktuell, rechter Op. +1, Ergebnis +2:
`>>[-]<< [[-] > [- > + <] <] >>`

■ Logisches Oder: Transferiere beide Operanden zur Ergebnisvariable.

■ Logisches Nicht: Setze Ergebnisvariable auf 1. Dekrementiere Ergebnisvariable in einem *If*-Stament, das die Eingangsvariable abfragt.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Vergleiche

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Vergleiche
 - Vergleich zweier Zellen a und b :

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Vergleiche
 - Vergleich zweier Zellen a und b :
 - Kopiere a und b nach a' und b'

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Vergleiche
 - Vergleich zweier Zellen a und b :
 - Kopiere a und b nach a' und b'
 - Berechne $a - b$ und $b' - a'$

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Vergleiche

■ Vergleich zweier Zellen a und b :

- Kopiere a und b nach a' und b'
- Berechne $a - b$ und $b' - a'$
- Falls beide Ergebnisse Null sind, waren die Werte gleich.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Vergleiche

■ Vergleich zweier Zellen a und b :

- Kopiere a und b nach a' und b'
- Berechne $a - b$ und $b' - a'$

■ Falls beide Ergebnisse Null sind, waren die Werte gleich.

■ Einfacher Vergleich mit einer Konstanten:

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Vergleiche

■ Vergleich zweier Zellen a und b :

- Kopiere a und b nach a' und b'
- Berechne $a - b$ und $b' - a'$
- Falls beide Ergebnisse Null sind, waren die Werte gleich.

■ Einfacher Vergleich mit einer Konstanten:

- Initialisiere Hilfsvariable mit 1, ziehe die Konstante mit Folge von Minuszeichen ab, starte Schleife, dekrementiere Hilfsvariable, dann addiere auf ursprüngliche Zelle die Konstante drauf, danach setze auf Null.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Vergleiche

■ Vergleich zweier Zellen a und b :

- Kopiere a und b nach a' und b'
- Berechne $a - b$ und $b' - a'$
- Falls beide Ergebnisse Null sind, waren die Werte gleich.

■ Einfacher Vergleich mit einer Konstanten:

- Initialisiere Hilfsvariable mit 1, ziehe die Konstante mit Folge von Minuszeichen ab, starte Schleife, dekrementiere Hilfsvariable, dann addiere auf ursprüngliche Zelle die Konstante drauf, danach setze auf Null.
- $>[-]+< \dots - [>-< +\dots + [-]] >$

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



■ Vergleiche

■ Vergleich zweier Zellen a und b :

- Kopiere a und b nach a' und b'
- Berechne $a - b$ und $b' - a'$
- Falls beide Ergebnisse Null sind, waren die Werte gleich.

■ Einfacher Vergleich mit einer Konstanten:

- Initialisiere Hilfsvariable mit 1, ziehe die Konstante mit Folge von Minuszeichen ab, starte Schleife, dekrementiere Hilfsvariable, dann addiere auf ursprüngliche Zelle die Konstante drauf, danach setze auf Null.
- `>[-]+< -...- [>-< +...+ [-]] >`

■ Weitere Tipps: http://www.iwriteiam.nl/Ha_bf_intro.html

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Vergleiche
 - Vergleich zweier Zellen a und b :
 - Kopiere a und b nach a' und b'
 - Berechne $a - b$ und $b' - a'$
 - Falls beide Ergebnisse Null sind, waren die Werte gleich.
 - Einfacher Vergleich mit einer Konstanten:
 - Initialisiere Hilfsvariable mit 1, ziehe die Konstante mit Folge von Minuszeichen ab, starte Schleife, dekrementiere Hilfsvariable, dann addiere auf ursprüngliche Zelle die Konstante drauf, danach setze auf Null.
 - `>[-]+< -...- [>-< +...+ [-]] >`
- Weitere Tipps: http://www.iwriteiam.nl/Ha_bf_intro.html
- Auf dieser Basis können alle Konstrukte nach Brainf*ck übersetzt werden.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Vergleiche
 - Vergleich zweier Zellen a und b :
 - Kopiere a und b nach a' und b'
 - Berechne $a - b$ und $b' - a'$
 - Falls beide Ergebnisse Null sind, waren die Werte gleich.
 - Einfacher Vergleich mit einer Konstanten:
 - Initialisiere Hilfsvariable mit 1, ziehe die Konstante mit Folge von Minuszeichen ab, starte Schleife, dekrementiere Hilfsvariable, dann addiere auf ursprüngliche Zelle die Konstante drauf, danach setze auf Null.
 - `>[-]+< -... - [>-< +...+ [-]] >`
- Weitere Tipps: http://www.iwriteiam.nl/Ha_bf_intro.html
- Auf dieser Basis können alle Konstrukte nach Brainf*ck übersetzt werden.
- Vergleichbar mit Compilerbau.

Motivation

Program-
miersprache

Beispiele

Schleife

Hello World

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Semantik

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung

Semantik (von 1993)



Short: 240 byte compiler. Fun, with src. OS 2.0
Uploader: umueller amiga physik unizh ch
Type: dev/lang
Architecture: m68k-amigaos

The brainfuck compiler knows the following instructions:

Cmd	Effect
---	-----
+	Increases element under pointer
-	Decreases element under pointer
>	Increases pointer
<	Decreases pointer
[Starts loop, flag under pointer
]	Indicates end of loop
.	Outputs ASCII code under pointer
,	Reads char and stores ASCII under ptr

Who can program anything useful with it? :)

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung

Semantik (von 1993)



Short: 240 byte compiler. Fun, with src. OS 2.0
Uploader: umueller amiga physik unizh ch
Type: dev/lang
Architecture: m68k-amigaos

The brainfuck compiler knows the following instructions:

Cmd	Effect
---	-----
+	Increases element under pointer
-	Decreases element under pointer
>	Increases pointer
<	Decreases pointer
[Starts loop, flag under pointer
]	Indicates end of loop
.	Outputs ASCII code under pointer
,	Reads char and stores ASCII under ptr

Who can program anything useful with it? :)

Leider lässt die Angabe der Semantik einige Fragen offen.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- 1 **Zellgröße:** In der ursprünglichen Implementierung 1 Byte (= 8 Bits) entsprechend den Zahlen von 0...255. Andere Implementierungen benutzen z.T. größere Zellen.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- 1 **Zellgröße**: In der ursprünglichen Implementierung 1 Byte (= 8 Bits) entsprechend den Zahlen von 0...255. Andere Implementierungen benutzen z.T. größere Zellen.
- 2 **Anzahl der Datenzellen**: Ursprünglich 30000. Aber auch andere Größen sind üblich. Manche Implementierungen benutzen nur 9999, andere erweitern die Liste auch dynamisch, manchmal sogar ins Negative hinein.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- 1 **Zellgröße**: In der ursprünglichen Implementierung 1 Byte (= 8 Bits) entsprechend den Zahlen von 0...255. Andere Implementierungen benutzen z.T. größere Zellen.
- 2 **Anzahl der Datenzellen**: Ursprünglich 30000. Aber auch andere Größen sind üblich. Manche Implementierungen benutzen nur 9999, andere erweitern die Liste auch dynamisch, manchmal sogar ins Negative hinein.
- 3 **Zeilenendezeichen**: `\n` (Unix) oder `\r\n` (Windows)? Meist die Unix-Konvention.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- 1 **Zellgröße**: In der ursprünglichen Implementierung 1 Byte (= 8 Bits) entsprechend den Zahlen von 0...255. Andere Implementierungen benutzen z.T. größere Zellen.
- 2 **Anzahl der Datenzellen**: Ursprünglich 30000. Aber auch andere Größen sind üblich. Manche Implementierungen benutzen nur 9999, andere erweitern die Liste auch dynamisch, manchmal sogar ins Negative hinein.
- 3 **Zeilenendezeichen**: `\n` (Unix) oder `\r\n` (Windows)? Meist die Unix-Konvention.
- 4 **Dateiende** (EOF): Beim Ausführen von , wird die Zelle entweder auf 0 gesetzt, nicht geändert, oder (bei Implementierungen mit größeren Zellen) auf -1 gesetzt.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- 1 Zellgröße:** In der ursprünglichen Implementierung 1 Byte (= 8 Bits) entsprechend den Zahlen von 0...255. Andere Implementierungen benutzen z.T. größere Zellen.
- 2 Anzahl der Datenzellen:** Ursprünglich 30000. Aber auch andere Größen sind üblich. Manche Implementierungen benutzen nur 9999, andere erweitern die Liste auch dynamisch, manchmal sogar ins Negative hinein.
- 3 Zeilenendezeichen:** `\n` (Unix) oder `\r\n` (Windows)? Meist die Unix-Konvention.
- 4 Dateiende (EOF):** Beim Ausführen von , wird die Zelle entweder auf 0 gesetzt, nicht geändert, oder (bei Implementierungen mit größeren Zellen) auf -1 gesetzt.
- 5 Unbalancierte Klammern:** Das Verhalten ist nicht spezifiziert!

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen
Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Die meisten Programmiersprachen haben ähnliche Probleme.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Die meisten Programmiersprachen haben ähnliche Probleme.
- Speziell der Bereich der darstellbaren Zahlen ist ein Problem.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Die meisten Programmiersprachen haben ähnliche Probleme.
- Speziell der Bereich der darstellbaren Zahlen ist ein Problem.
- Oft wird festgelegt, dass es **Implementierungs-abhängige** Größen und Werte gibt (z.B. maximale Größe einer Zahl).

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Die meisten Programmiersprachen haben ähnliche Probleme.
- Speziell der Bereich der darstellbaren Zahlen ist ein Problem.
- Oft wird festgelegt, dass es **Implementierungs-abhängige** Größen und Werte gibt (z.B. maximale Größe einer Zahl).
- Oft gibt es **Freiheiten bei der Implementierung** (z.B. Reihenfolge der Auswertung in Ausdrücken).

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Die meisten Programmiersprachen haben ähnliche Probleme.
- Speziell der Bereich der darstellbaren Zahlen ist ein Problem.
- Oft wird festgelegt, dass es **Implementierungs-abhängige** Größen und Werte gibt (z.B. maximale Größe einer Zahl).
- Oft gibt es **Freiheiten bei der Implementierung** (z.B. Reihenfolge der Auswertung in Ausdrücken).
- Außerdem gibt es immer Dinge, die außerhalb der Spezifikation einer Sprache liegen (z.B. Verhalten bei unbalancierten Klammern).

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Die meisten Programmiersprachen haben ähnliche Probleme.
- Speziell der Bereich der darstellbaren Zahlen ist ein Problem.
- Oft wird festgelegt, dass es **Implementierungs-abhängige** Größen und Werte gibt (z.B. maximale Größe einer Zahl).
- Oft gibt es **Freiheiten bei der Implementierung** (z.B. Reihenfolge der Auswertung in Ausdrücken).
- Außerdem gibt es immer Dinge, die außerhalb der Spezifikation einer Sprache liegen (z.B. Verhalten bei unbalancierten Klammern).
- Hier ist das **Verhalten undefiniert**, aber idealerweise wird eine Fehlermeldung erzeugt (statt erraticem Verhalten).

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen

Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



Brainf*ck-Programme, die **portabel** (d.h., auf möglichst vielen Implementierungen lauffähig) sind, müssen ein paar Konventionen einhalten:

- Bei Zellgröße nur ein Byte annehmen. Ggfs. sogar nur den Bereich von 0–127 nutzen, da es bei einer vorzeichenbehafteten Darstellung einen arithmetischen Überlauf geben könnte!

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen
Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



Brainf*ck-Programme, die **portabel** (d.h., auf möglichst vielen Implementierungen lauffähig) sind, müssen ein paar Konventionen einhalten:

- Bei Zellgröße nur ein Byte annehmen. Ggfs. sogar nur den Bereich von 0–127 nutzen, da es bei einer vorzeichenbehafteten Darstellung einen arithmetischen Überlauf geben könnte!
- Zur Erkennung der EOF-Marke sollte vor dem Lesen die Zelle auf Null gesetzt werden. So ist die Zelle auf jeden Fall 0, falls das Eingabeende erreicht wird.

Motivation

Program-
miersprache

Beispiele

Semantik

Offene Fragen
Portabilität

Interpreter-
Design

Ausblick

Zusammen-
fassung



Interpreter-Design

Motivation

Program-
miersprache

Beispiele

Semantik

**Interpreter-
Design**

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell eines Brainf*ck Programms: $src \in \mathbb{N} \leftrightarrow B$

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell eines Brainfuck Programms: $src \in \mathbb{N} \leftrightarrow B$
- Operationen: nur Lesen, an beliebiger Stelle

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell eines Brainfuck Programms: $src \in \mathbb{N} \leftrightarrow B$
- Operationen: nur Lesen, an beliebiger Stelle
- Was ist der geeignete Datentyp dafür?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell eines Brainfuck Programms: $src \in \mathbb{N} \leftrightarrow B$
- Operationen: nur Lesen, an beliebiger Stelle
- Was ist der geeignete Datentyp dafür?
 - Einfache Lösung: **String**! Aber Schleifen etwas umständlich und ineffizient.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell eines Brainf*ck Programms: $src \in \mathbb{N} \leftrightarrow B$
- Operationen: nur Lesen, an beliebiger Stelle
- Was ist der geeignete Datentyp dafür?
 - Einfache Lösung: **String**! Aber Schleifen etwas umständlich und ineffizient.
 - Profi-Lösung: Rekursive Datenstruktur mit Schachtelung entsprechend der Klammerstruktur; dafür muss der String in eine passende interne Datenstruktur transformiert werden.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell der Brainf*ck Datenzellen: $data \in \mathbb{N} \rightarrow \mathbb{N}$

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell der Brainf*ck Datenzellen: $data \in \mathbb{N} \rightarrow \mathbb{N}$
- Operationen: Lesen, Schreiben an beliebigen Stellen, Initialisieren auf 0

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell der Brainf*ck Datenzellen: $data \in \mathbb{N} \rightarrow \mathbb{N}$
- Operationen: Lesen, Schreiben an beliebigen Stellen, Initialisieren auf 0
- Was ist der geeignete Datentyp dafür?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Modell der Brainf*ck Datenzellen: $data \in \mathbb{N} \rightarrow \mathbb{N}$
- Operationen: Lesen, Schreiben an beliebigen Stellen, Initialisieren auf 0
- Was ist der geeignete Datentyp dafür?
- Ein **Dictionary** passt am besten.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



Wir haben es mit drei Ein-/Ausgabeströmen zu tun:

- 1 Das Programm: einmal einlesen und dann verarbeiten.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



Wir haben es mit drei Ein-/Ausgabeströmen zu tun:

- 1 Das Programm: einmal einlesen und dann verarbeiten.
- 2 Eingabestrom: Datei oder Konsole.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



Wir haben es mit drei Ein-/Ausgabeströmen zu tun:

- 1 Das Programm: einmal einlesen und dann verarbeiten.
- 2 Eingabestrom: Datei oder Konsole.
- 3 Ausgabestrom: Datei oder Konsole.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



Wir haben es mit drei Ein-/Ausgabeströmen zu tun:

- 1 Das Programm: einmal einlesen und dann verarbeiten.
 - 2 Eingabestrom: Datei oder Konsole.
 - 3 Ausgabestrom: Datei oder Konsole.
- Das Modul `sys` stellt zwei `File` Objekte für die **Standardeingabe** und **Standardausgabe** zur Verfügung: `sys.stdin` und `sys.stdout`

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Falls ein Dateiname angegeben wurde, wird die dazugehörige Datei geöffnet.

bf.py: Open files

```
import sys

def open_files(srcfn : str, infn : Optional[str], outfn : Optional[str]):
    fin = open(infn)          if infn else sys.stdin
    fout = open(outfn, "w") if outfn else sys.stdout
    return(open(srcfn), fin, fout)
```

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



Wo können Fehler passieren?

- Dateifehler (Existenz/Lesen/(Über-)Schreiben)
- Sollten wir besser abfangen!
- Fehler beim Interpretieren des Programms (Teilen durch 0 usw.)
- Für die Fehlersuche bei der Entwicklung erst einmal nicht abfangen, später dann schon.
- Verletzung von Sprachregeln wie z.B. Nicht-ASCII-Zeichen > 127, oder unbalancierte Klammern.
- Wir definieren einen speziellen Ausnahmetyp.

Spezielle Exception

```
class BFEError(Exception):  
    pass
```

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen
I/O

**Ausnahmebehand-
lung**

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Main function

```
def bf(sfn : str, infn : Optional[str], outf : Optional[str]):  
  
    try:  
        (src,fin,fout) = open_files(sfn, infn, outf)  
        pass # TBI: Aufruf des Interpreters  
    except IOError as e:  
        print("I/O-Fehler:", e)  
    except BFEError as e:  
        print("Abbruch wegen BF-Inkompatibilität:",e)  
    except Exception as e:  
        print("Interner Interpreter-Fehler:", e)  
    finally:  
        fout.close()
```

- Hier gibt es noch ein/zwei Problemchen!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Main function

```
def bf(sfn : str, infn : Optional[str], outfn : Optional[str]):  
    fout = None  
    try:  
        (src,fin,fout) = open_files(sfn, infn, outfn)  
        pass # TBI: Aufruf des Interpreters  
    except IOError as e:  
        print("I/O-Fehler:", e)  
    except BFError as e:  
        print("Abbruch wegen BF-Inkompatibilität:",e)  
    except Exception as e:  
        print("Interner Interpreter-Fehler:", e)  
    finally:  
        if fout and outfn: fout.close()
```

- Hier gab es noch ein/zwei Problemchen!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf0.py

```
def bfinterpret(src : str, fin, fout):
    # Program counter points into source text
    pc = 0
    # data pointer
    ptr = 0
    # data cells are stored in a dict
    data = dict()

    while pc < len(src):
        if src[pc] == '>':
            ptr += 1
        elif src[pc] == '+':
            data[ptr] = data.get(ptr,0) + 1
        elif ...
        pc += 1
```

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



- Sehr lange if-else-Anweisungen sind schwer lesbar, insbesondere wenn die Anweisungsblöcke groß werden.
- Alternative: jede Bedingung ruft eine Funktion auf oder ...
- Ein Dictionary ordnet jedem BF-Befehl (als Schlüssel) eine Funktion zu, die Semantik des Befehls implementiert.
- Die Fallunterscheidung geschieht durch den Zugriff aufs Dictionary.
- Wesentliche Vereinfachung: die Hauptfunktion passt auf eine Folie!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Main interpreter loop

```
def bfinterpret(srctext : str, fin, fout):  
    pc = 0  
    ptr = 0  
    data = dict()  
    while pc < len(srctext):  
        (pc, ptr) = instr.get(srctext[pc],noop)(pc,  
                                           ptr, srctext, data, fin, fout)  
        pc += 1
```

Es fehlt noch ein `dict instr`, das mit jeder BF-Instruktion eine Funktion assoziiert, die 6 Parameter besitzt (den Zustandsraum) und die ein Paar `(pc, ptr)` zurückgibt.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



```
bf.py: instr_table
```

```
instr = { '<': left, '>': right,  
         '+': incr, '-': decr,  
         '.': ch_out, ',': ch_in,  
         '[': beginloop, ']': endloop }
```

- Diese Tabelle kann erst **nach den Funktionsdefinitionen** angelegt werden.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung

Die einfachen Fälle (1)



bf.py: Simple cases

```
def noop(pc, ptr, src, data, fin, fout):
    return(pc, ptr)

def left(pc, ptr, src, data, fin, fout):
    return(pc, ptr - 1 if ptr > 0 else 0)

def right(pc, ptr, src, data, fin, fout):
    return(pc, ptr + 1)
```

Beachte: Der `pc` wird in der Hauptschleife erhöht!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Simple cases

```
def incr(pc, ptr, src, data, fin, fout):
    data[ptr] = data.get(ptr,0) + 1
    return(pc, ptr)

def decr(pc, ptr, src, data, fin, fout):
    vold = data.get(ptr,0)
    data[ptr] = vold - 1 if vold > 0 else 0
    return(pc, ptr)
```

Beachte: Es sind beliebig viele Zellen erlaubt.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: I/O

```
def ch_in(pc, ptr, src, data, fin, fout):
    ch = fin.read(1)
    if ch:
        data[ptr] = ord(ch)
    return(pc, ptr)

def ch_out(pc, ptr, src, data, fin, fout):
    print(chr(data.get(ptr,0)), end='', file=fout)
    return(pc, ptr)
```

Was passiert, wenn die Ein- oder Ausgabe kein gültiges ASCII-Zeichen ist?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



```
bf.py: I/O
```

```
def ch_in(pc, ptr, src, data, fin, fout):
    ch = fin.read(1)
    if ch:
        data[ptr] = ord(ch)
        if data[ptr] > 127:
            raise BFEError("Non-ASCII-Zeichen gelesen")
    return(pc, ptr)

def ch_out(pc, ptr, src, data, fin, fout):
    if data.get(ptr,0) > 127:
        raise BFEError("Ausgabe eines Non-ASCII-Zeichen")
    print(chr(data.get(ptr,0)), end='', file=fout)
    return(pc, ptr)
```

- Motivation
- Programmiersprache
- Beispiele
- Semantik
- Interpreter-Design
- Datenstrukturen
- I/O
- Ausnahmebehandlung
- Hauptfunktion
- Fallunterscheidung
- Einfache Fälle
- I/O
- Schleifen
- Ausblick
- Zusammenfassung



bf.py: Loop begin

```
def beginloop(pc, ptr, src, data, fin, fout):
    if data.get(ptr,0):
        return (pc, ptr)
    loop = 1;
    while loop > 0:
        pc += 1
        if src[pc] == ']':
            loop -= 1
        elif src[pc] == '[':
            loop += 1
    return(pc, ptr)
```

Frage: Was passiert bei unbalancierten Klammern?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Loop begin

```
def beginloop(pc, ptr, src, data, fin, fout):
    if data.get(ptr,0): return (pc, ptr)
    loop = 1;
    while loop > 0:
        pc += 1
        if pc >= len(src):
            raise BFError("Kein passendes ']' gefunden")
        if src[pc] == ']':
            loop -= 1
        elif src[pc] == '[':
            loop += 1
    return(pc, ptr)
```

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Loop end

```
def endloop(pc, ptr, src, data, fin, fout):
    loop = 1;
    while loop > 0:
        pc -= 1
        if src[pc] == ']':
            loop += 1
        elif src[pc] == '[':
            loop -= 1
    return(pc - 1, ptr)
```

Frage: Was passiert bei unbalancierten Klammern?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



bf.py: Loop end

```
def endloop(pc, ptr, src, data, fin, fout):
    loop = 1;
    while loop > 0:
        pc -= 1
        if pc < 0:
            raise BFEError("Kein passendes '[' gefunden")
        if src[pc] == ']':
            loop += 1
        elif src[pc] == '[':
            loop -= 1
    return(pc - 1, ptr)
```

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Datenstrukturen

I/O

Ausnahmebehand-
lung

Hauptfunktion

Fallunterscheidung

Einfache Fälle

I/O

Schleifen

Ausblick

Zusammen-
fassung



Ausblick

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung

Was nun?



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.
- BF ist **Turing-vollständig**, d.h. prinzipiell können alle Algorithmen in BF implementiert werden!

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.
- BF ist **Turing-vollständig**, d.h. prinzipiell können alle Algorithmen in BF implementiert werden!
- Z.B. ein Programm zum Berechnen aller Werte der **Fakultätsfunktion**.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.
- BF ist **Turing-vollständig**, d.h. prinzipiell können alle Algorithmen in BF implementiert werden!
- Z.B. ein Programm zum Berechnen aller Werte der **Fakultätsfunktion**.
- Z.B. ein **Adventure-Spiel**.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.
- BF ist **Turing-vollständig**, d.h. prinzipiell können alle Algorithmen in BF implementiert werden!
- Z.B. ein Programm zum Berechnen aller Werte der **Fakultätsfunktion**.
- Z.B. ein **Adventure-Spiel**.
- Z.B. ein Programm, das BF-Programme interpretiert, also ein **BF-Interpreter geschrieben in BF**.

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.
- BF ist **Turing-vollständig**, d.h. prinzipiell können alle Algorithmen in BF implementiert werden!
- Z.B. ein Programm zum Berechnen aller Werte der **Fakultätsfunktion**.
- Z.B. ein **Adventure-Spiel**.
- Z.B. ein Programm, das BF-Programme interpretiert, also ein **BF-Interpreter geschrieben in BF**.
- Wie wäre es mit einem Brainf*ck→Python Compiler (in Python oder Brainf*ck)?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



- Wir können BF-Programme schreiben und vom Interpreter ausführen lassen!
- Zum Beispiel das **Hello-World**-Programm.
- BF ist **Turing-vollständig**, d.h. prinzipiell können alle Algorithmen in BF implementiert werden!
- Z.B. ein Programm zum Berechnen aller Werte der **Fakultätsfunktion**.
- Z.B. ein **Adventure-Spiel**.
- Z.B. ein Programm, das BF-Programme interpretiert, also ein **BF-Interpreter geschrieben in BF**.
- Wie wäre es mit einem Brainf*ck→Python Compiler (in Python oder Brainf*ck)?
- Oder umgekehrt?

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung



Zusammenfassung

- Motivation
- Program-
miersprache
- Beispiele
- Semantik
- Interpreter-
Design
- Ausblick
- Zusammen-
fassung**



- Brainf*ck ist eine **minimale, Turing-vollständige** Programmiersprache.
- Es ist relativ einfach, für diese Sprache einen Interpreter zu schreiben.
- Wir könnten auch einen Interpreter für Brainf*ck in Brainf*ck schreiben.
- Ähnlich können wir einen Interpreter für Python in Python schreiben.
- Diese Art von Interpreter ist die Basis des PyPy Projekts, eine alternative Python-Implementierung, die oft schneller als CPython läuft.

<https://en.wikipedia.org/wiki/PyPy>

Motivation

Program-
miersprache

Beispiele

Semantik

Interpreter-
Design

Ausblick

Zusammen-
fassung