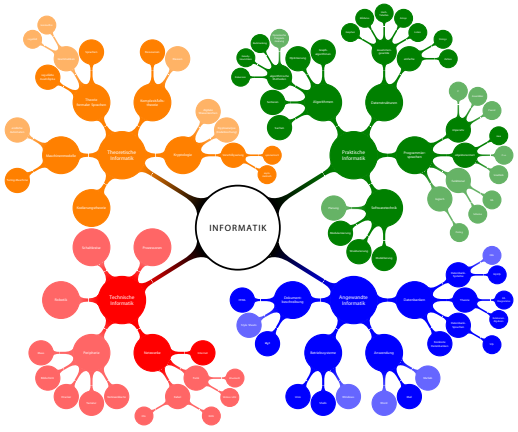
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 9

Imperative Programmierung

Tu was ich sage! (Leider nicht: Tu was ich meine.)

Vorlesung Einführung in die Informatik 1 vom 21. November 2013 von Till Tantau



Lernziele von Kapitel 9

1. Zuweisungen verstehen und benutzen können
2. Grundlegende Steuerungsanweisungen verstehen

Gliederung von Kapitel 9

Wiederholung:
Der Arbeitsfluss beim Lösen eines Problems.

1. Ausgangspunkt ist ein *Problem*.
2. Um sich klar zu machen, was man will, erstellt man eine *Spezifikation*.
3. Dann entwickelt man eine *Lösungsidee*.
4. Diese verfeinert man zu einem *Lösungsalgorithmus*.
5. Diesen implementiert man als ein *Programm* in einer Hochsprache.
6. Auf dieses wendet man einen *Übersetzer* an, um eine Instruktionsfolge zu erhalten.

Programmiersprachen unterscheiden sich in verschiedenen Dimensionen.

Erste Dimension: Maschinennähe

Leistung

Visuelle Sprachen

Java

C

Assembler

Maschinensprache

Programmiersprachen unterscheiden sich in verschiedenen Dimensionen.

Zweite Dimension: Paradigma

- Funktionale Sprachen (wenig verbreitet):
  Lisp, Scheme, ML
- Logische Sprachen (sehr wenig verbreitet):
  Prolog, Datalog
- Objektorientierte Sprachen (sehr verbreitet):
  Java, C++, C#, Modula-3, Smalltalk
- Imperative Sprachen (sehr verbreitet):
  C, Pascal, Maschinensprachen.

Die Evolution von Programmierern.

Grundschule

```
10 PRINT "Hallo␣Welt."
20 END
```

Die Evolution von Programmierern.

Gymnasium

```
program Hello(input, output)
begin
  writeln('Hello World')
end.
```

Die Evolution von Programmierern.

Uni, erstes Semester

```
#include <stdio.h>
void main(void)
{
    char *message[] = {"Hello␣", "World"};
    int i;

    for(i = 0; i < 2; ++i)
      printf("%s", message[i]);
    printf("\n");
}
```

Die Evolution von Programmierern.

Professioneller Programmierer

```
#include <iostream>
#include <iomanip>
class string
{
  private:
    int size;
    char *ptr;
    string() : size(0), ptr(new char[1]) { ptr[0] = 0; }
    string(const string &s) : size(s.size)
    {
      ptr = new char[size + 1];
      strcpy(ptr, s.ptr);
    }
    ~string() { delete [] ptr; }
    friend ostream &operator <<(ostream &, const string &);
};

ostream &operator<<(ostream &stream, const string &s)
{ return(stream << s.ptr); }

int main()
{
    string str;
    str = "Hello World";
    cout << str << endl;
    return(0);
}
```

Die Evolution von Programmierern.

Hacker (Anfänger)

```
#!/usr/local/bin/perl
$msg="Hello, world.\n";
if ($#ARGV >= 0) {
  while(defined($arg=shift(@ARGV))) {
    $outfilename = $arg;
    open(FILE, ">" . $outfilename) ||
      die "Can't write $arg: $!\n";
    print (FILE $msg);
    close(FILE) || die "Can't close $arg: $";
  }
} else {
  print ($msg);
}
1;
```

Die Evolution von Programmierern.

Hacker (Profi)

```
> cc -o a.out ~/src/misc/hw/hw.c
> a.out
```

Die Evolution von Programmierern.

Hacker (Guru)

```
> echo "Hello, world."
```

Die Evolution von Programmierern.

Junior-Professor

```
> mail -s "Hallo␣Welt." stockhus@uni-luebeck.de
Koenntest Du mir bis naechste Woche ein
Programm schreiben, das "Hallo␣Welt"
ausgibt? Danke.
^D
```

Die Evolution von Programmierern.

W2-Professor

```
> mail mitarbeiter@theory.uni-luebeck.de
Ich brauche bis zur 14-Uhr-Vorlesung
ein Hallo-Welt-Programm.
^D
```

Die Evolution von Programmierern.

W3-Professor

Sehr geehrte Damen und Herren,

im Anhang finden Sie den Verlängerungsantrag für das Projekt »Initiale Maschine-Mensch-Kommunikation«.

Wir bitten um Verlängerung um weitere zwei Jahre und Aufstockung um eine weitere E13 Stelle.

Mit freundlichen Grüßen
Ich

Variablen dienen zur Speicherung von Werten.

- Ähnlich wie in der Mathematik stehen *Variablen* für veränderliche Werte.
- *Anders* als in der Mathematik können Variablen im Computer ihren Wert *tatsächlich ändern*.
- Variablen haben einen *Namen* (englisch *identifier*), der in Java aus Buchstaben und Zahlen bestehen darf.
- Beispiel: Java-Variablennamen sind `i`, `index1`, `mein_toller_Variablenname`.

Ausdrücke bestehen aus der Anwendung von Funktionen auf Variablen und Konstanten.

- Ein *Ausdruck* (englisch *expression*) kombiniert den Wert von verschiedenen Variablen zu einem neuen Wert.
- Beispiel: Der Ausdruck $x + y$ gibt den Wert der Summe der Variablen x und y an.
- Beispiel: Der Ausdruck $2 \cdot (i + 1)$ gibt den doppelten Wert des Nachfolgers von i an.
- Beispiel: Der Ausdruck $\mathrm{pow}(x, 2)$ gibt denselben Wert wie $x \cdot x$ an.

Boolesche Ausdrücke können wahr oder falsch sein.

- Eine besondere Art Ausdrücke sind *boolesche* Ausdrücke.
- Wie anderen Ausdrücke auch kombinieren sie Variablen und Konstanten zu einem neuen Wert -- nun allerdings zu einem der beiden Werte `true` oder `false`.
- Beispiel: Der Ausdruck $x > 5$ ist genau dann gleich `true`, wenn x größer als fünf ist.
- Beispiel: Der Ausdruck $x < y \cdot 2$ ist genau dann gleich `true`, wenn x kleiner dem Doppelten von y ist.
- Beispiel: Der Ausdruck $2 \leq x \wedge x \leq 5$ ist genau dann gleich `true`, wenn x zwischen 2 und 5 liegt.

Schreibweisen für Operatoren zur Bildung von booleschen Ausdrücken.

Mathematisch/ Pseudocode	Pascal, Modula, Oberon	C, C++, C#
$x = y$	`x = y`	`x == y`
$x \neq y$	`x <> y`	`x != y`
$x \leq y$	`x <= y`	`x <= y`
$x \geq y$	`x >= y`	`x >= y`
$x < y$	`x < y`	`x < y`
$x > y$	`x > y`	`x > y`

Zuweisung dienen dazu, Variablen einen neuen Wert zu geben.

- Eine *Zuweisung* ist der elementarste Befehl, den es in imperativen Programmen gibt.
- Eine Zuweisung weist den Computer an, einer Variablen den Wert zu geben, den ein Ausdruck gerade hat.

Beispiel

Die Zuweisung $x \leftarrow x + y$ besagt, dass x *nach* Ausführung der Zuweisung den Wert haben soll, den $x + y$ *vorher* hatte.

5-Minuten-Aufgabe

Zur Übung

Finden Sie heraus, welche Werte die Variablen x und y nach Ausführung folgender Zuweisungen haben:

1 $x \leftarrow 5$
2 $y \leftarrow 10$
3 $z \leftarrow x + y$
4 $x \leftarrow z$
5 $z \leftarrow y$
6 $y \leftarrow z \cdot z - y$
7 $x \leftarrow z + x + y$

5-Minuten-Aufgabe

Zur Übung

Finden Sie heraus, welche Werte die Variablen x und y nach Ausführung folgender Zuweisungen haben:

1 $y \leftarrow x + y$
2 $x \leftarrow y - x$
3 $y \leftarrow y - x$

Die Komposition bezeichnet einfach die Hintereinanderausführung von Befehlen.

- Imperative Programme bestehen aus Folgen von Befehlen.
- Solche Folgen entstehen durch *Komposition* von Befehlen.
- Da diese Steuerungsanweisung so wichtig und einfach ist, ist sie syntaktisch sehr einfach:
  - In Pascal deutet ein Semikolon die Komposition von Befehlen an.
  - In Java kann man die Zuweisungen einfach hintereinander weg schreiben.

Die Alternativen-Steuerungsanweisung erlaubt es, in Abhängigkeit eines Ausdrucks unterschiedliche Dinge zu tun.

- Die Alternative besteht aus zwei bis drei Teilen:
  1. Einer *Bedingung*.
  2. Einem *Then-Zweig*.
  3. Einem *Else-Zweig* (optional).
- Die Bedingung ist ein boolescher Ausdruck. Dieser wird als erstes ausgewertet:
  - Wenn er wahr ist, werden die Anweisungen im Then-Zweig befolgt.
  - Wenn er falsch ist, werden die Anweisungen im Else-Zweig befolgt. (Falls dieser fehlt, geht es einfach mit dem nächsten Befehl weiter.)

Schreibweise der Alternativen in Pseudo-Code.


```
1 if Bedingung then
2     Anweisung des Then-Zweigs
3 else
4     Anweisung des Else-Zweigs
```

5-Minuten-Aufgabe

Zur Übung

Welche Werte habe die Variablen am Ende von folgendem Programm?

```
1  x ← 8
2  y ← 5
3  if x < y then
4      x ← y
5  else
6      z ← x
7      x ← y
8      y ← z
```

Die While-Steuerungsanweisung erlaubt es, eine Befehlsfolge wiederholt auszuführen.

- Die While-Schleife besteht aus zwei Teilen:
  1. Einer *Bedingung*.
  2. Einem *Körper*.
- Die Bedingung ist ein boolescher Ausdruck. Dieser wird bei jedem Schleifendurchlauf am Anfang ausgewertet.
- Wenn die Bedingung wahr ist, wird der Körper ausgewertet, danach wieder die Bedingung, dann wieder der Körper und so fort.
- Wenn die Bedingung falsch ist, wird die Schleife beendet.

Schreibweise der Schleife in Pseudo-Code.


```
1 while Bedingung do
2     Anweisung des Körpers
```

5-Minuten-Aufgabe

Zur Übung

Welchen Wert hat *sum* am Ende von folgendem Programm?

```
1  n ← 0
2  sum ← 0
3  while n ≤1000 do
4      sum ← sum + n
5      n ← n + 1
```

Zusammenfassung

Variablen

Ein »Computer-Variable« liegt irgendwo im Speicher und *ändert ihren Wert* nach jeder Zuweisung.

Zuweisung

Ein *Zuweisung* ist ein Befehl der Form $v \leftarrow a$. Hierzu wird zuerst der *Ausdruck* a ausgewertet und das Resultat danach in den Speicherbereich geschrieben, den die Variable v belegt.

Alternative

Eine *Alternative* hat die Form

```
1 if Bedingung then
2     Anweisung des Then-Zweigs
3 else
4     Anweisung des Else-Zweigs
```

Schleife

Eine *Schleife* hat die Form

```
1 while Bedingung do
2     Körper
```