Vorlesung an der

Berufsakademie Oldenburg

Unterrichtseinheit 10

Variablen
(Hamster-Modell)

Dr. Dietrich Boles

Gliederung

- Motivation
- Variablen
  - Definition
  - Initialisierung
- Ausdrücke
  - Motivation
  - Definition
  - Operatoren
  - Zuweisung
  - Operatoreneigenschaften
- Ort der Variablendefinition
- Beispiel
- Gültigkeitsbereich und Lebensdauer
- Codekonventionen
- Aufgaben

Motivation

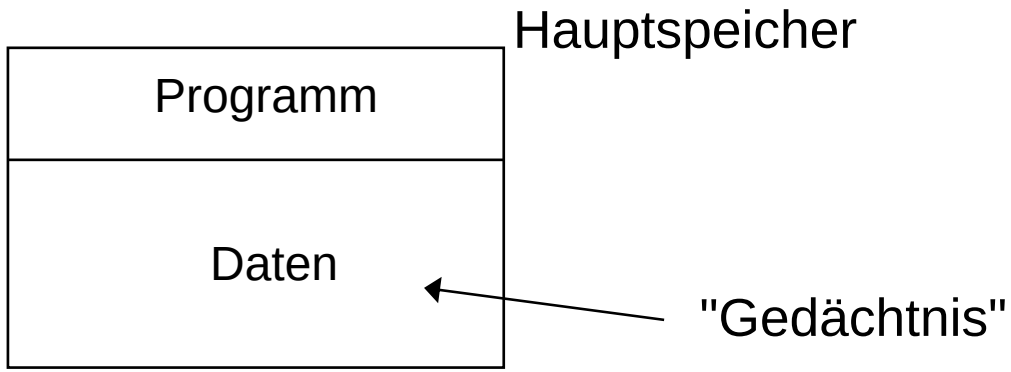
- Motivation:
  - Der Hamster soll bis zur nächsten Wand laufen, dabei alle Körner einsammeln, die er findet, und an der Wand genauso viele Körner ablegen, wie er eingesammelt hat.
- Notwendige Voraussetzungen:
  - Gedächtnis (⟶ Speicher)
  - Zählen/Rechnen/Vergleichen (⟶ Operationen)
- Programm:

```
*1:"gefundene Anzahl ist 0";
while (vornFrei()) {
    vor();
    while (kornDa()) {
        nimm();
        *2:"erhöhe gefundene Anzahl um 1";
  }
}
while ( *3:"gefundene Anzahl größer als 0 ist" ) {
    gib();
    *4:"erniedrige gefundene Anzahl um 1";
}
```

Variablen / Definition (1)

- Notwendig für *[1]:
  - Reservierung von Speicherplatz (→ Variablendefinition)
  - Festlegung eines Namens (→ Variablendeklaration)
  - Festlegung des Typs (→ Datentypen)
  - Initialer Wert (→ Initialisierung)
  - hier: `int anzahl = 0;`

- Variablendeklaration/-definition:
  - Deklaration: Bekanntgabe eines Namens für den Speicherbereich an den Compiler
  - Definition: Reservierung von Platz im Hauptspeicher



Variablen / Definition (2)

Syntax:

```
<Variablendefinition> ::= <Typ>
                          <Bezeichner> "=" <Ausdruck>
                          {"," <Bezeichner> "=" <Ausdruck>}
                          ";"

<Typ>                 ::= "int"
```

Semantik:

Es wird Speicherplatz zum Speichern einer Zahl angelegt und initialisiert.

Beispiele:

```
int anzahl = 0;
int koelnischWasser = 4711;
int eins = 1, zwei = 2, drei = 3;
```

Variablen / Initialisierung

- Initialisierung von Variablen
  - implizit durch Default-Wert 0
  - explizit mit Hilfe von Literalen bzw. Ausdrücken
  - Empfehlung: Variablen immer explizit initialisieren!

- Literale: Typ-spezifische Konstanten

  - **`int`**:

    Zeichenketten aus
    - dezimalen **`29`**
    - oktalen (führende 0): **`035`**
    - hexadezimalen Ziffern (führendes 0x): **`0x1D`**

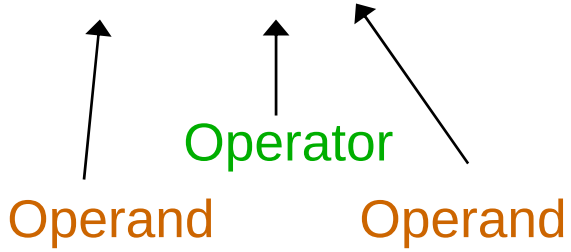
Ausdrücke / Motivation

- Notwendig für `*2` , `*3` und `*4` im Beispiel auf Folie 3:
  - Operationen (→ Mathematik)
  - Operatoren
  - Operanden

- Sinn und Zweck: Berechnung neuer Werte aus alten Werten

- Beispiel:

```
int anzahl = 0;
anzahl = anzahl + 1;
```



Ausdrücke / Definition

- Ausdruck:
  - Verarbeitungsvorschrift, deren Ausführung einen **Wert** liefert
  - entsteht, indem **Operanden** mit **Operatoren** verknüpft werden

Syntax:

```
<Ausdruck> ::=   <Literal>
               | <Variablenname>
               | "(" <Ausdruck> ")"
               | <unär-Op> <Ausdruck>
               | <Ausdruck> <binär-Op> <Ausdruck>
```

Semantik:

Literal: liefert Wert des Literals

Variablenname: liefert den aktuell in der Variablen gespeicherten Wert

Ausdrücke / Operatoren (1)

geliefert wird die berechnete Zahl

+:	positives Vorzeichen	`+9876`
-:	negatives Vorzeichen	`-2345`
+:	Addition	`12 + 98 (= 110)`
-:	Subtraktion	`12 - 98 (= -86)`
*****:	Multiplikation	`6 * 5 (= 30)`
/:	Ganzzahl-Division	`7 / 3 (= 2)`
%:	Restbildung (modulo)	`7 % 3 (= 1)`

Ausdrücke / Operatoren (2)

geliefert wird boolescher Wert (`true`, `false`)

==:	Gleichheit	`4 == 5`	`(= false)`
!=:	Ungleichheit	`6 != 7`	`(= true)`
<:	Kleiner	`-2 < 1`	`(= true)`
<=:	Kleiner-Gleich	`3 <= 3`	`(= true)`
>:	Größer	`3 > 3`	`(= false)`
>=:	Größer-Gleich	`1 >= -1`	`(= true)`

Ausdrücke / Zuweisung (1)

Motivation:

Änderung von Variablenwerten

Syntax:

```
<Zuweisung> ::= <Variablenname> "=" <Ausdruck> ";"
```

Semantik:

Der Ausdruck wird berechnet und der berechnete Wert
in der Variablen gespeichert.
Der alte Wert der Variablen geht verloren.

Beispiele:

```
int schritte = 3; // Initialisierung
schritte = 5;     // Zuweisung
schritte = schritte + 1;
schritte = 5 * (schritte -2);
```

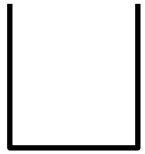
Ausdrücke / Zuweisung (2)

Schema:

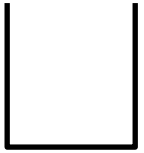
```
int zahl = 4;

int number = 12;

zahl = zahl + number / 5;
```



zahl



number

Ausdrücke / Zuweisung (3)

Abkürzungen:

```
i++              ⟷   i = i + 1    ⟷    ++i
i--              ⟷   i = i - 1    ⟷    --i
i += <expr>      ⟷   i = i + (<expr>)
i -= <expr>      ⟷   i = i - (<expr>)
i *= <expr>      ⟷   i = i * (<expr>)
i /= <expr>      ⟷   i = i / (<expr>)
i %= <expr>      ⟷   i = i % (<expr>)
```

Ausdrücke / Operatoren / Eigenschaften

- **Präzedenz** (absteigend):
  - Postfix-Operatoren `++, --`
  - Multiplikationsoperatoren `*, /, %`
  - Additionsoperatoren `+, -`
  - Gleichheitsoperatoren `==, !=`
  - Zuweisungsoperatoren `=, +=, *=, ...`
  - Abänderung der Präzedenz mit Hilfe von Klammern
- **Assoziativität**:
  - wichtig bei Operatoren gleicher Präzedenz
  - alle unären Operatoren sind rechts-assoziativ
  - alle binären Operatoren außer den Zuweisungsop. sind links-assoziativ
  - Zuweisungsoperatoren sind rechts-assoziativ

Ort der Variablendefinition

zwei Varianten von Variablendefinitionen:

als Definition (**globale Variable**)

```
int schritte = 0;

void main() {
    while (vornFrei()) {
        vor();
        schritte++;
    }
}
```

als Anweisung (**lokale Variable**)

```
void main() {
    int schritte = 0;
    while (vornFrei()) {
        vor();
        schritte++;
    }
}
```

Beispiel

- Motivation:

  Der Hamster soll bis zur nächsten Wand laufen, dabei alle Körner einsammeln, die er findet, und an der Wand genauso viele Körner ablegen, wie er eingesammelt hat.

- Programm:

```
void main() {
    int anzahl = 0;
    while (vornFrei()) {
        vor();
        while (kornDa()) {
            nimm();
            anzahl = anzahl + 1;
        }
    }
    while (anzahl > 0) {
        gib();
        anzahl = anzahl - 1;
    }
}
```

Gültigkeitsbereich und Lebensdauer (1)

- Gültigkeitsbereich einer Variablen:
  Der Teil eines Programms, in dem eine Variable genutzt werden kann.

- bei lokalen Variablen:
  - beschränkt sich auf den Block, in dem die Variable definiert wurde.
  - beginnt ab der Stelle der Definition.

- bei globalen Variablen:
  - umfasst das gesamte Programm.
  - Ausnahmen: eigener Initialisierungsausdruck und Initialisierungsausdrücke vorher definierter globaler Variablen.

- Allgemein:
  - Im Gültigkeitsbereich einer Variablen darf keine neue Variable gleichen Namens definiert werden; Ausnahme: Gleichnamige globale und lokale Variablen sind erlaubt (lokale überladen globale Variablen)

Gültigkeitsbereich und Lebensdauer (2)

- Lebensdauer einer Variablen:

  Zeitspanne, während der Speicherplatz für eine Variable reserviert ist.

- bei lokalen Variablen:
  - beginnt bei ihrer Definition und endet nach der vollständigen Abarbeitung des Blockes, in dem sie definiert wurde

- bei globalen Variablen:
  - umfasst die gesamte Ausführungszeit des Programms

Gültigkeitsbereich und Lebensdauer (3)

- Aufgabe:

  Der Hamster soll bis zur nächsten Wand laufen, dabei alle Körner einsammeln, die er findet, und an der Wand genauso viele Körner ablegen, wie er eingesammelt hat.

- **Korrektes Programm** mit globaler Variable **`anzahl`**:

```
int anzahl = 0;

void main() {
  sammle();
  while (vornFrei()) {
    vor();
    sammle();
  }
  legeAb();
}
```

```
void sammle() {
  while (kornDa()) {
    nimm();
    anzahl = anzahl + 1;
  }
}

void legeAb() {
  while (anzahl > 0) {
    gib();
    anzahl = anzahl - 1;
  }
}
```

Gültigkeitsbereich und Lebensdauer (4)

- Aufgabe:

  Der Hamster soll bis zur nächsten Wand laufen, dabei alle Körner einsammeln, die er findet, und an der Wand genauso viele Körner ablegen, wie er eingesammelt hat.

- **Fehlerhaftes Programm** mit zwei lokalen Variablen **`anzahl`**:

```
void main() {
  sammle();
  while (vornFrei()) {
    vor();
    sammle();
  }
  legeAb();
}
```

```
void sammle() {
  int anzahl = 0;
  while (kornDa()) {
    nimm();
    anzahl = anzahl + 1;
} }

void legeAb() {
  int anzahl = 0;
  while (anzahl > 0) {
    gib();
    anzahl = anzahl - 1;
} }
```

Codekonventionen

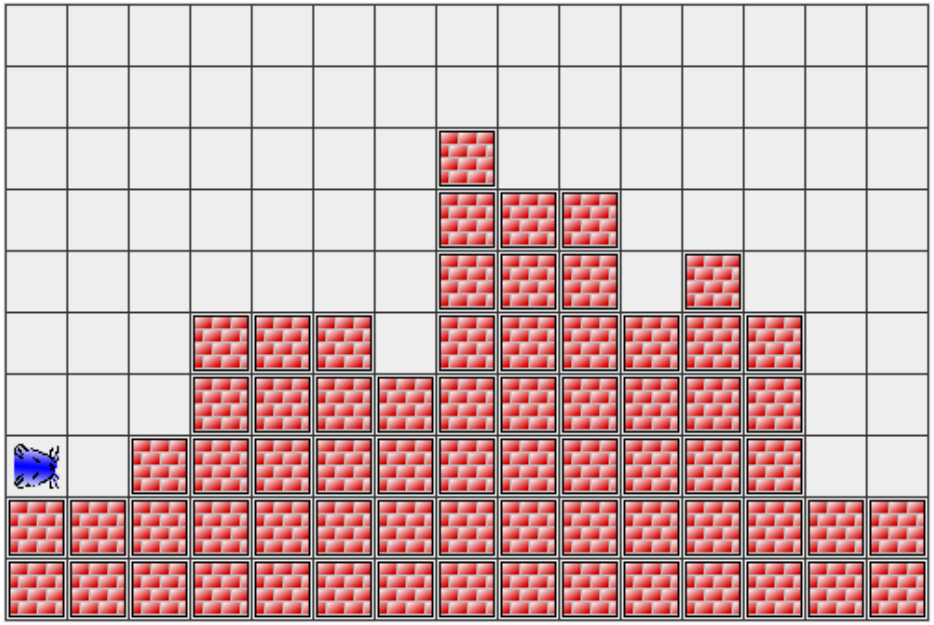
- Variablenname: Anfangsbuchstabe klein; Anfangsbuchstaben neuer Wortbestandteile groß
- Variablen möglichst einzeln deklarieren
- immer explizit initialisieren
- keine gleichnamigen globalen und lokalen Variablen verwenden
- vor und hinter binären Operatoren ein Leerzeichen

Aufgabe 1

Aufgabe:

Der Hamster soll das Gebirge übersteigen.

Beispiellandschaft:



Aufgabe 2

Aufgabe:

Der Hamster soll die Körnerzahlen der oberen Reihen addieren und in der unteren Reihe ablegen.

Beispiellandschaft: