

## 4. Variable, Lebensdauer

Themen dieses Kapitels:

- Variablenbegriff und Zuweisung
- unterschiedliche Lebensdauer von Variablen
- Laufzeitkeller als Speicherstruktur für Variablen in Aufrufen

# Variable in imperativen Sprachen

**Variable:** wird **im Programm beschrieben**, z. B. durch Deklaration (**statisch**),  
wird **bei der Ausführung** im Speicher **erzeugt** und verwendet (**dynamisch**),  
wird charakterisiert durch das Tripel (**Name**, **Speicherstelle**, **Wert**).

Einem **Namen im Programm** werden (bei der Ausführung) eine oder mehrere **Stellen im Speicher** zugeordnet.

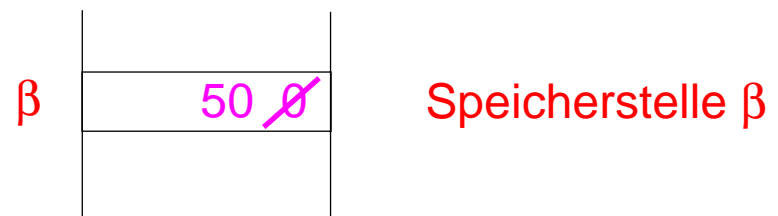
Das Ausführen von **Zuweisungen** ändert den **Wert der Variablen (Inhalt der Speicherstelle)**.  
Bei der Ausführung eines imperativen Programms wird so der **Programmzustand** verändert.

Der Deklaration einer **globalen (static) Variable** ist genau eine Stelle zugeordnet.  
Der Deklaration einer **lokalen Variablen einer Funktion** wird bei jeder Ausführung eines Aufrufes eine neue Stelle zugeordnet.

im Programm:

```
int betrag = 0;
...
betrag = 50;
```

im Speicher bei der Ausführung:



## Veränderliche und unveränderliche Variable

In **imperativen Sprachen** kann der Wert einer Variablen grundsätzlich **durch Ausführen von Zuweisungen verändert** werden.

```
int betrag = 0;
...
betrag = 50;
```

In manchen **imperativen Sprachen**, wie Java, kann für bestimmte Variable **verboten** werden, nach ihrer Initialisierung an sie **zuzuweisen**.

```
final int hekto = 100;
```

In **funktionalen Sprachen** wird bei der Erzeugung einer **Variablen** ihr **Wert unveränderlich** festgelegt.

```
val sechzehn = (sqr 4);
```

In **mathematischen Formeln** wird ein **Wert unveränderlich an** den Namen einer **Variablen gebunden**. (Die Formel kann mit verschiedenen solchen Name-Wert-Bindungen ausgewertet werden.)

$$\forall x, y \in \mathbb{R}: y = 2 * x - 1$$

definiert eine Gerade im  $\mathbb{R}^2$

# Zuweisung

**Zuweisung: LinkeSeite = RechteSeite;**

**Ausführen einer Zuweisung:**

1. **Auswerten der linken Seite;**  
muss die **Stelle einer Variablen** liefern.
2. **Auswerten der rechten Seite**  
liefert einen **Wert**.  
**In Ausdrücken stehen Namen von Variablen für ihren Wert**, d. h. es wird implizit eine **Inhaltsoperation** ausgeführt.
3. Der **Wert der Variablen** aus (1) **wird** durch den Wert aus (2) **ersetzt**.

**Beispiel**

im Programm:

```
b = 42;
c = b + 1;
i = 3;
a[i] = c;
```

im Speicher:

Speicherstelle zu

b	42
c	43
i	3
a	
a[3]	43

## Stellen als Werte von Variablen

In objektorientierten Sprachen, wie Java oder C++, liefert die Ausführung von `new C(...)` die Stelle (Referenz) eines im Speicher erzeugten Objektes. Sie kann in Variablen gespeichert werden.

Java:

```
Circles cir =
    new Circles(0, 1.0);
x = cir.getRadius();
```

C++:

```
Circles *cir =
    new Circles(0, 1.0);
x = cir->getRadius();
```

In C können Pointer-Variable Stellen als Werte haben (wie in C++). Die Ausführung von `malloc (sizeof(Circle))` liefert die Stelle (Referenz) eines im Speicher erzeugten Objektes.

C:

```
Circles *cir =
    malloc(sizeof(Circle));
cir->radius = 1.0;
```

Der Ausdruck `&i` liefert die Stelle der deklarierten Variable `i`, d. h. der `&`-Operator **unterdrückt die implizite Inhaltsoperation**. Der Ausdruck `*i` **bewirkt eine Inhaltsoperation** - zusätzlich zu der impliziten.

```
int i = 5, j = 0;
int *p = &i;
j = *p + 1;
p = &i;
```

# Lebensdauer von Variablen im Speicher

**Lebensdauer:** Zeit von der Bildung (Allokation) bis zur Vernichtung (Deallokation) des Speichers einer Variablen. Begriff der **dynamischen Semantik!**

Art der Variablen	Lebensdauer ist die Ausführung ...	Unterbringung im Speicher
globale Variable Klassenvariable	... des gesamten Programms	globaler Speicher
Parametervariable, lokale Variable	... eines Aufrufes	Laufzeitkeller
Objektvariable	... des Programms von der Erzeugung bis zur Vernichtung des Objekts	Halde, ggf. mit Speicher- bereinigung

Variable mit gleicher Lebensdauer werden zu **Speicherblöcken** zusammengefasst. (Bei Sprachen mit geschachtelten Funktionen kommen auch Funktionsrepräsentanten dazu.)

## Speicherblock für

- Klassenvariable einer Klasse
- einen Aufruf mit den Parametervariablen und lokalen Variablen
- ein Objekt einer Klasse mit seinen Objektvariablen

# Laufzeitkeller

Der **Laufzeitkeller** enthält für jeden noch nicht beendeten Aufruf einen Speicherblock (**Schachtel**, activation record) mit Speicher für Parametervariable und lokale Variable. Bei **Aufruf** wird eine **Schachtel** gekellert, bei **Beenden des Aufrufes** entkellert.

## Programm mit Funktionen (Methoden)

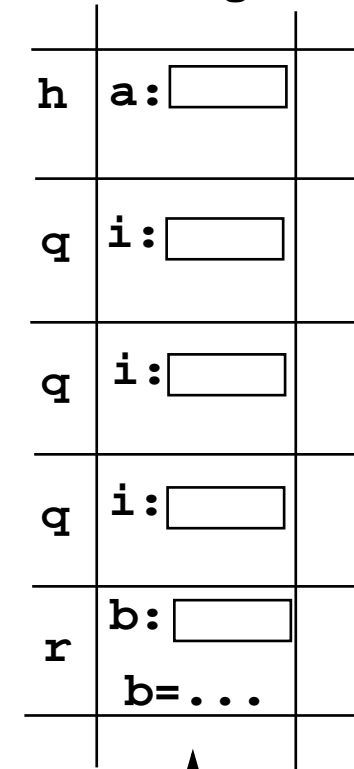
```

h
┌ float a;
│
│ q();
└

q
┌ int i;
│ if (...) q();
│ r();
└

r
┌ int b;
│
│ b=...;
└
  
```

## Laufzeitkeller bei der Aufruffolge h, q, q, q, r



↑  
kellern, entkellern

# Laufzeitkeller bei geschachtelten Funktionen

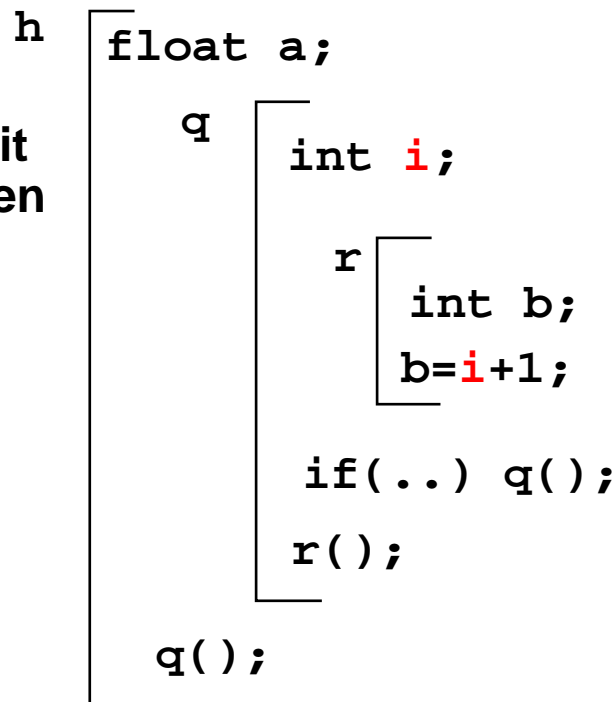
Bei der Auswertung von Ausdrücken kann auf Variablen aus der **Umgebung** zugegriffen werden. Das sind die Speicherblöcke zu den Programmstrukturen, die den Ausdruck umfassen.

in Pascal, Modula-2, in funktionalen Sprachen: geschachtelte Funktionen

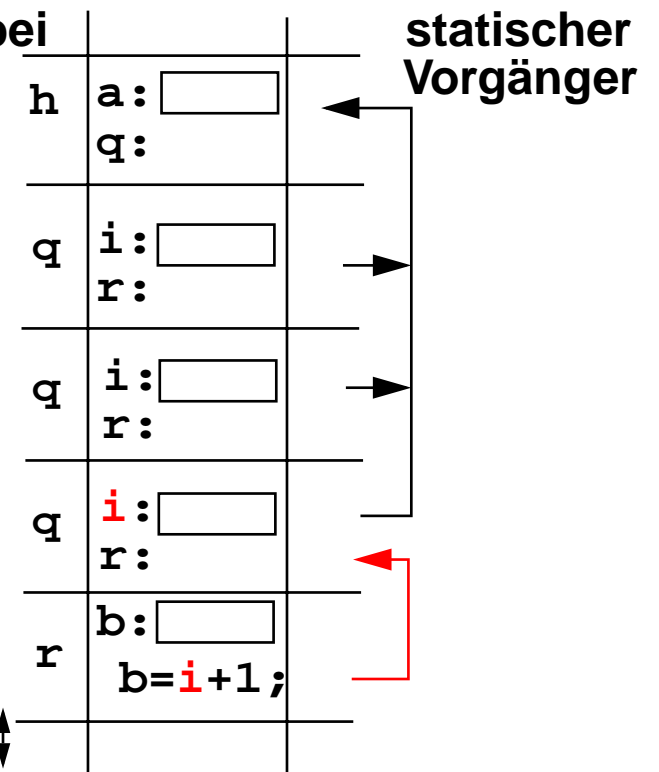
in Java: Methoden in Klassen, geschachtelte Klassen

Im **Laufzeitkeller** wird die **aktuelle Umgebung** repräsentiert durch die aktuelle Schachtel und die Schachteln entlang der Kette der **statischen Vorgänger**. Der statische Vorgänger zeigt auf die Schachtel, die die Definition der aufgerufenen Funktion enthält.

Programm mit geschachtelten Funktionen



Laufzeitkeller bei Aufruf von r



# Zusammenfassung zum Kapitel 4

Mit den Vorlesungen und Übungen zu Kapitel 4 sollen Sie nun Folgendes verstanden haben:

- Variablenbegriff und Zuweisung
- Zusammenhang zwischen Lebensdauer von Variablen und ihrer Speicherung
- Prinzip des Laufzeitkellers
- Besonderheiten des Laufzeitkellers bei geschachtelten Funktionen