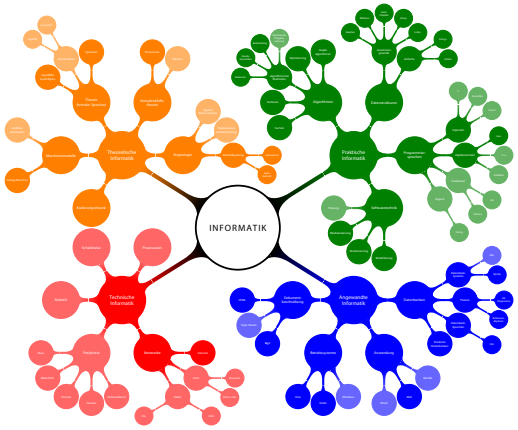
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 14

Scoping

Globalisierung und Subsidiaritätsprinzip

Vorlesung Einführung in die Informatik 1 vom 10. Dezember 2013 von Till Tantau



Lernziele von Kapitel 14

1. Scoping verstehen und anwenden können.
2. Die Speicherung von Variablen verstehen.

Gliederung von Kapitel 14

Manche Namen sind besser als andere.

(Alle Tiere sind gleich. Aber manche sind gleicher als andere.)

- Warum gibt es so viele Peter Müller und Jan Fischer?
- Warum gibt es so wenige Cornelia Schmalz-Jacobsen und Sabine Leutheusser-Schnarrenberger?

Manche Namen sind besser als andere.

(Alle Tiere sind gleich. Aber manche sind gleicher als andere.)

- Warum gibt es so viele Peter Müller und Jan Fischer?
- Warum gibt es so wenige Cornelia Schmalz-Jacobsen und Sabine Leutheusser-Schnarrenberger?

Moral

Gute Bezeichner sind selten!

Gute Bezeichner müssen viel leisten.

Gute Bezeichner sind

- kurz,
- aussagekräftig und
- leicht zu merken.

Bezeichner aus einem Programm.

Gute Bezeichner sind:

- `sum`,
- `length`,
- `List`,
- `velocity`.

Nicht ganz so gute Bezeichner sind

- `j`,
- `tikznumberofcurrentchild`.

Schlechte Bezeicher sind

- `tikz@collect@children@cchar`,
- `tikz@p@c@n@c@at`.

Das Problem des Name-Clashes.

Ein *Name-Clash* liegt vor, wenn derselbe Bezeichner für unterschiedliche Dinge verwendet werden soll:

```
...
int sum_n = 0, sun_m = 0;

int i;
for (i=0; i<=n; i = i+1) {
  sum_n = sum_n+i;
}

int i;   // Verboten!
for (i=0; i<=m; i = i+1) {
  sum_m = sum_m+i;
}
```

Einfache Lösungen der Name-Clashes.

- Man kann eine der Variablen umbenenen:

```
int i;
for (i=0; i<=n; i = i+1) {
  sum_n = sum_n+i;
}

int j;
for (j=0; j<=m; j = j+1) {
  sum_m = sum_m+j;
}
```

Nachteil: Umbenennungen sind fehlerträchtig und nicht immer möglich.

- Man kann die Variable einfach zweimal benutzen:

```
int i;
for (i=0; i<=n; i = i+1) {
  sum_n = sum_n+i;
}

for (i=0; i<=m; i = i+1) {
  sum_m = sum_m+i;
}
```

Nachteil: Variablen werden nicht dort deklariert, wo sie benutzt werden.

Name-Clashes kommen auch auf anderen Ebenen vor.


```
// SAP:
class List {
...
}

...

// Software AG:
class List {
...
}
```

Variablen sollten nicht zu lange Speicher belegen.

- Variablen belegen unter Umständen sehr viel Speicher (Beispiel: Variable für ein Bild).
- Wird eine Variable nicht mehr benutzt, so sollte der Speicher wiederverwendet werden.

Woher weiß nun aber der Übersetzer, wann eine Variable nicht mehr gebraucht wird?

Die Lebensdauer zweier Variablen.


```
int i;
for (i=0; i<=n; i = i+1) {
  sum_n = sum_n+i;
}

// Ab hier wird i gar nicht mehr gebraucht.
// Aber woher soll der Übersetzer das wissen?

int j;
for (j=0; j<=m; j = j+1) {
  sum_m = sum_m+j;
}
```

Die Lösung aller Probleme: Scopes.

- Ein *Scope* ist ein bestimmter Bereich eines Programms,
- der in Java in der Regel durch geschweifte Klammern eingeschlossen wird.
- Solche Scopes lassen sich schachteln.

Beispiel eines Programms mit mehreren Scopes.


```
class Hello
{ // Start von Scope 1

  public static void main (String[] args)
  { // Start von Scope 2
    int i = 0;

    while (i < 5)
    { // Start von Scope 3
      i = i + 1;
    } // Ende von Scope 3


    while (i > 0)
    { // Start von Scope 4
      i = i - 1;
    } // Ende von Scope 4

  } // Ende von Scope 2

} // Ende von Scope 1
```

Variablen sind nur innerhalb ihres Scopes gültig.


```
...
int sum_n = 0, sun_m = 0;

{
  int i;
  for (i=0; i<=n; i = i+1) {
    sum_n = sum_n+i;
  }
}

// Hier gibt es kein i mehr!

{
  int i;  // Ok!
  for (i=0; i<=m; i = i+1) {
    sum_m = sum_m+i;
  }
}
```

5-Minuten-Aufgabe

Zur Übung

Welche Variablen sind an den Orten A, B und C gültig?

```
int n, m;
{
  for (int i = 0; i < n; i = i+1) {
    n = n+m;
  }

  // Ort A

  int j;
  {
    for (j = 0; j < n; j = j+1) {
      m = m+n;
    }
  }
  // Ort B
}
// Ort C
```

Wer ist gemeint?

»Die Bank steht in einem Park.«

Die Verdeckung von Bezeichnern.

- Bezeichner innerhalb eines Scopes *verdecken* gleiche Bezeichner weiter außen.
- Es gilt also immer der Bezeichner, der am weitesten innen deklariert wurde.

5-Minuten-Aufgabe

Zur Übung

Welchen Werte haben `i` und `j` am Ende?

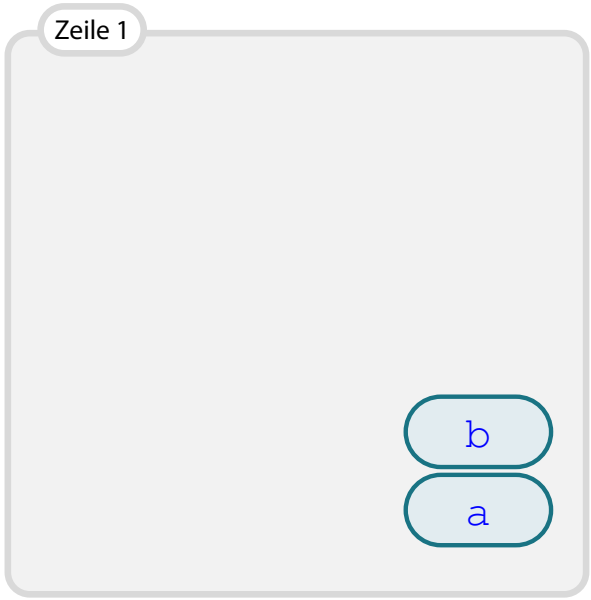
```
int i = 0;
int j = 0;
{
  int i = 5;
  j = 0;
  {
    int j = i;
    i = j+1;
    j = i-1;
  }
  j = i;
}
```

Scopes helfen dem Übersetzer bei der Speicherreservierung.

- Am Anfang eines Scopes merkt sich der Übersetzer, wie viel Speicherplatz bereits verbraucht ist.
- Wird dann eine neue Variable deklariert, so steigt der Speicherverbrauch.
- Am Ende eines Scopes kann aller Speicher freigegeben werden ab der gemerkten Stelle.

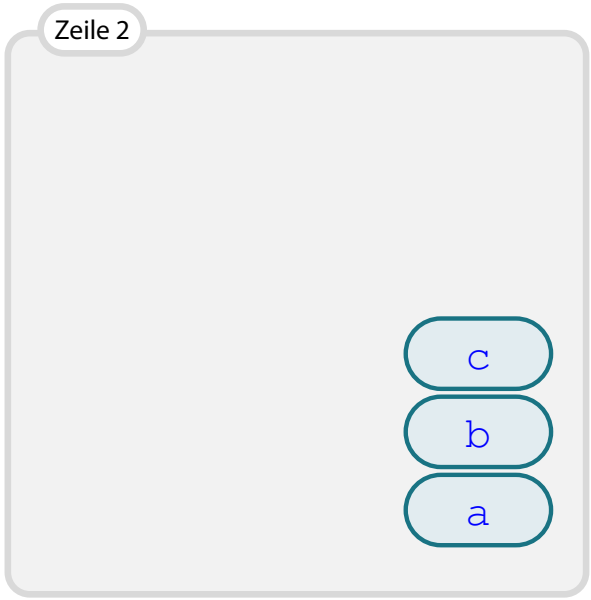
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



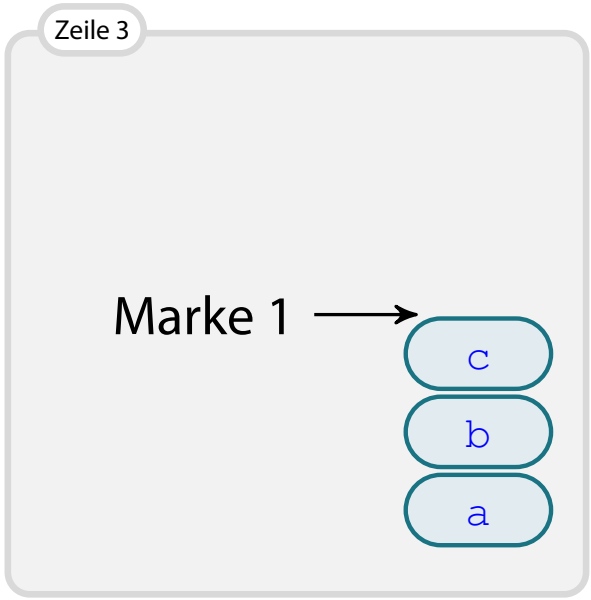
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



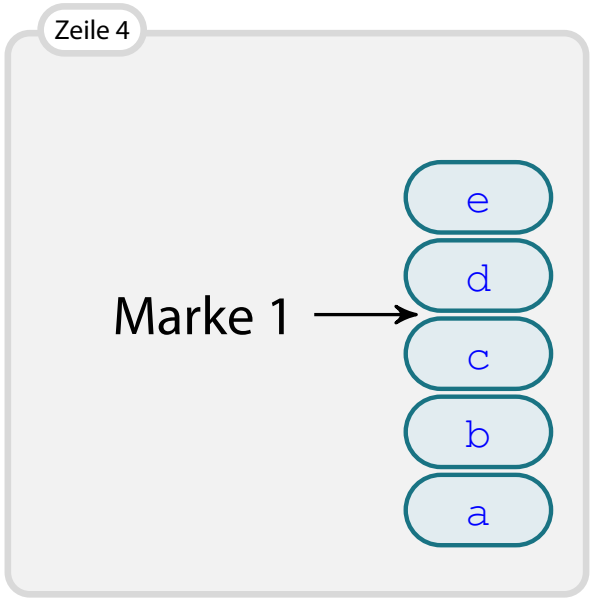
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



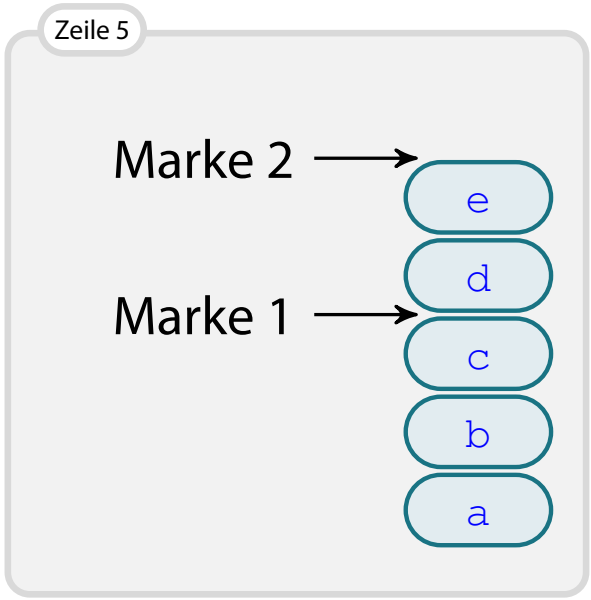
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e; // Zeile 4
  {         // Zeile 5
    int f; // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



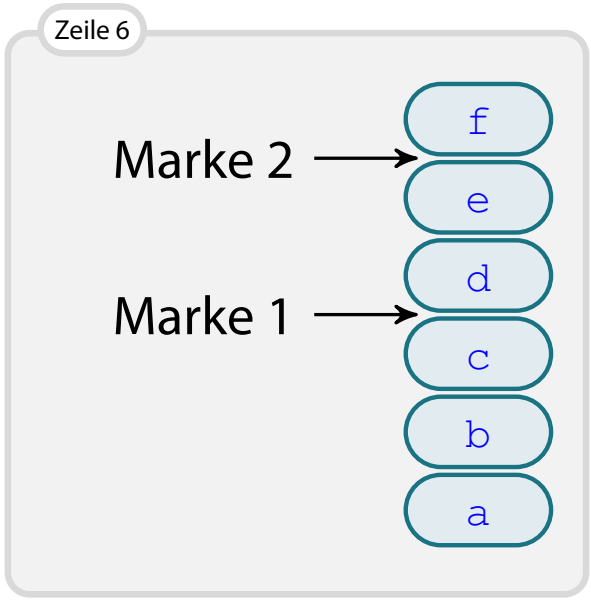
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



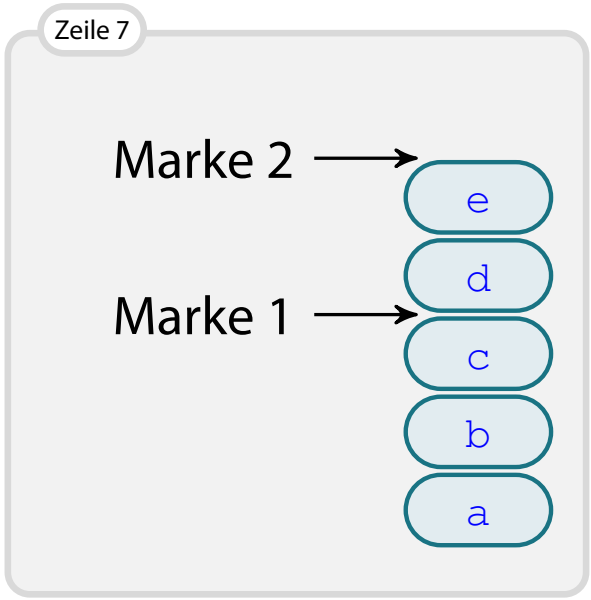
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



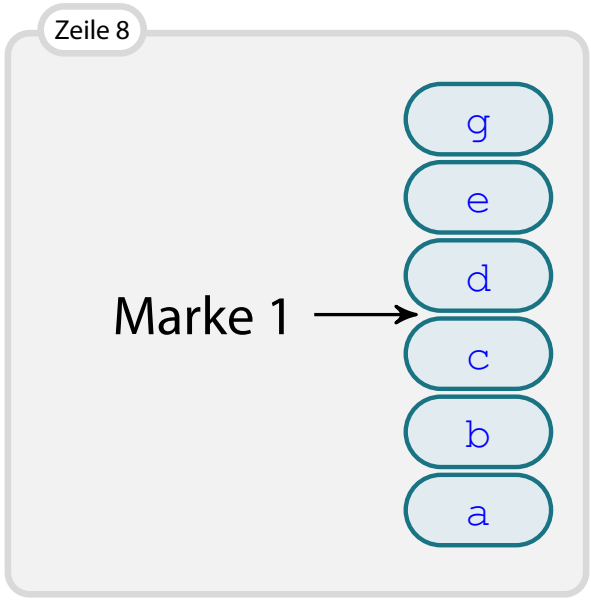
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



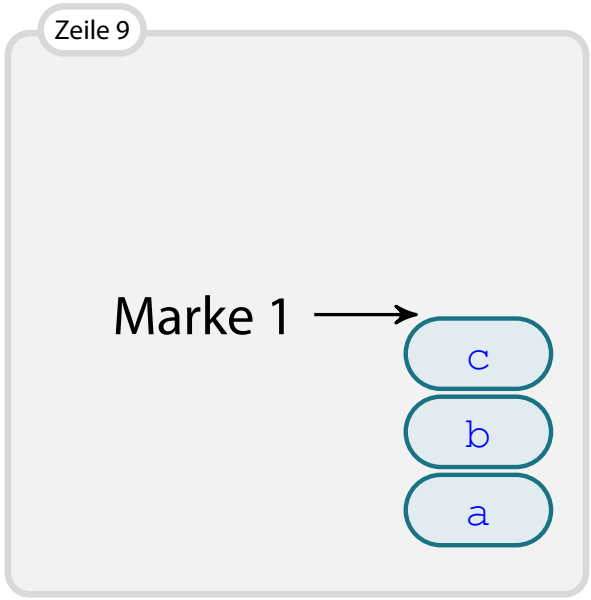
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



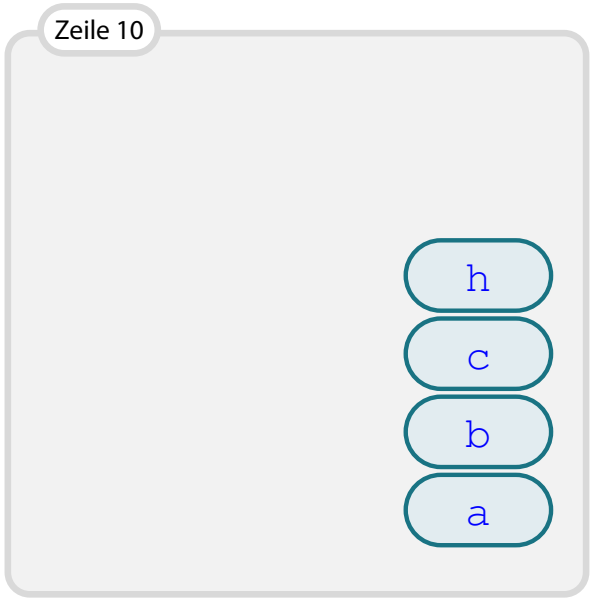
Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



Speicherbelegung während des Ablaufs eines Programms.


```
int a, b;   // Zeile 1
int c;      // Zeile 2
{           // Zeile 3
  int d,e;  // Zeile 4
  {         // Zeile 5
    int f;  // Zeile 6
  }         // Zeile 7
  int g;    // Zeile 8
}           // Zeile 9
int h;      // Zeile 10
```



Syntax von Scopes

Scopes beginnen mit einer geschweiften Klammer und enden bei der zugehörigen schließenden geschweiften Klammer.

```
{
  int i;
  ...
}
// Wir sind jetzt außerhalb des Scopes und
// i ist damit verschwunden
```

Ausnahme: `for` eröffnet direkt einen Scope:

```
for (int i = 0; ...; ...) {
  ...
}
// Wir sind jetzt außerhalb des Scopes der For-Schleife
// und i ist damit verschwunden
```

Zusammenfassung

Die Regeln hinter dem Scoping

1. Variablen *existieren* nur innerhalb des Scopes, in dem sie deklariert wurden.
2. Eine Variable kann in einem Scope nur einmal deklariert werden; »Sub-Scopes« dürfen aber Variablen »lokal« neu deklarieren.
3. Gibt es eine Variable gleichen Names mehrmals, so ist immer nur die *innerste sichtbar.*

Merke

Am Ende eines Scopes »verschwinden« möglicherweise einige Variable, jedoch ändern die verbliebenen *nicht ihren Wert*.