universitätbonn

Parameterübergabe

Parameterübergabe

- Es gibt mehrere Verfahren für Substitution der aktuellen Parameter für die formalen Parameter
  - Werteaufruf (call by value)
  - Referenzaufruf (call by reference)
  - Namensaufruf (call by name)

Parameterübergabe

- Werteaufruf (call by value)
  - Aktueller Parameter ai wird ausgewertet
  - Wert wird formalem Parameter ei zugewiesen
    - ⇨ Entspricht einer Zuweisung ei=ai
    - ⇨ Grundlegender Mechanismus bei vielen Programmiersprachen
      - Java, C, C++
  - Das Unterprogramm kann also als Seiteneffekt nicht den aktuellen Parameter ändern

Parameterübergabe

- Referenzaufruf (call by reference)
  - Aktueller Parameter ist Variable (im Unterprogramm)
    - ⇨ Zulässig jeder Ausdruck, der einen Linkswert hat
      - Z.B. also auch indizierte Variable wie a[i+5]
    - ⇨ Linkswert wird für den Linkswert des aktuellen Parameters eingesetzt
    - ⇨ Formaler Parameter wird ein alias für den aktuellen Parameter
      - Wert des formalen Parameters wird also nicht kopiert, sondern formaler Parameter verweist auf Wert
  - Zuweisungen an den formalen Parameter ändern also auch Wert des aktuellen Parameters
  - In Java nicht direkt verwendet
    - ⇨ Für Objekte wird eine eingeschränkte Variante verwendet
      - Siehe unten

Parameterübergabe

- Referenzaufruf (Forts.))
  - Unterstützt in C++ oder Pascal („var“-Parameter)
  - Bei Referenzaufruf kann Kopieren großer Strukturen vermieden werden
    - ⇨ Etwa großer Arrays
    - ⇨ In C++ kann syntaktisch gekennzeichnet werden, ob aktueller Parameter im Unterprogramm geändert werden kann oder nicht
  - Kann in C simuliert werden
    - ⇨ Verwende Zeigervariable
      - Call-by-value der Zeigervariable px, die auf die eigentliche Variable x verweist, simuliert Referenzaufruf von x

Parameterübergabe

- Namensaufruf (call by name)
  - Aktueller Parameter ist beliebiger Ausdruck
    - ⇨ Wird ohne Auswertung für den formalen Parameter substituiert
  - Ausdruck wird im Kontext des Unterprogramms immer dort evaluiert, wo der formale Parameter vorkommt
    - ⇨ Verwendung im λ-Kalkül
      - Und funktionalen Sprachen wie etwa LISP
      - Auch in ALGOL 68 konnte Namensaufruf verwendet werden
- Wird von Java, C++, Pascal etc. nicht unterstützt

Parameterübergabe

- **Beispiel:** Effekte der Übergabemechanismen auf die Reihung a beim Prozedur-Aufruf p(a[i])

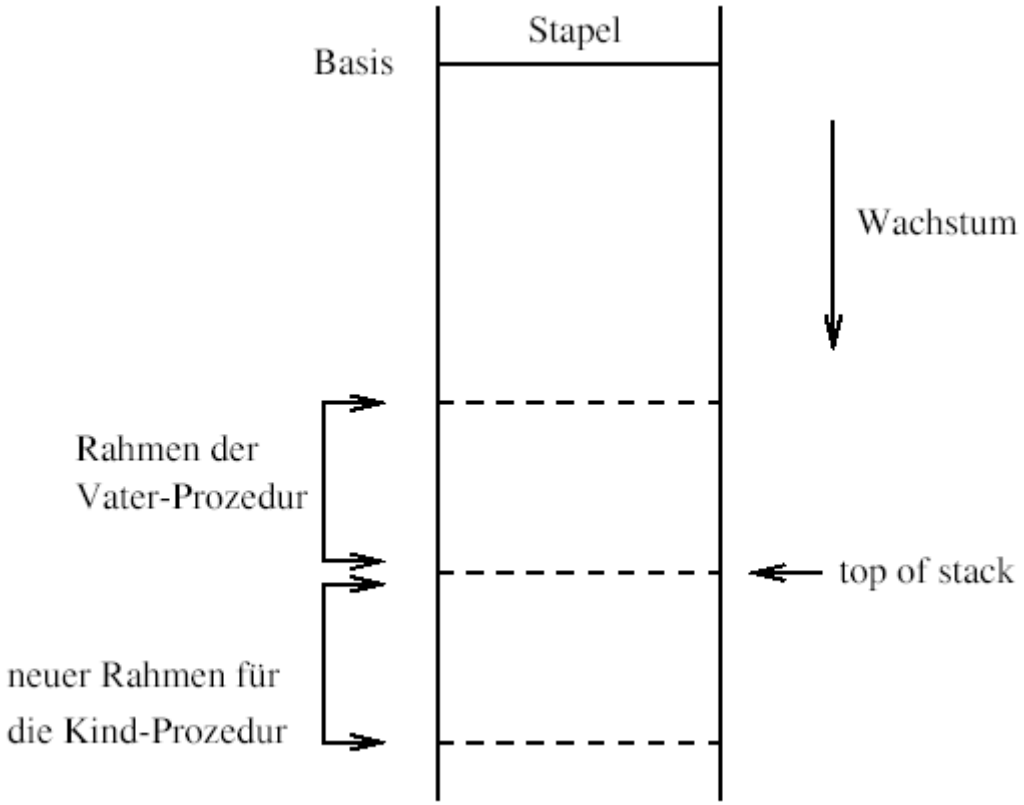
```
class Program {
  static int i=0;
  static int[] a=new int[] {10, 20};

  static void p(int x) {
    i=i+1; x=x+2;
  }

  public static void main(String[] argv) {
    p(a[i]);
    System.out.println(a[0]);
    System.out.println(a[1]);
  }
}
```

Werte-Aufruf: Zu Beginn von p wird implizit die Zuweisung x = 10 ausgeführt
Dann wird das Feld i in Program inkrementiert und die lokale Variable x in p um 2 erhöht, was auf a[0] keine Auswirkungen hat. Es wird 10 und 20 ausgegeben.

Referenz-Aufruf (in Java nicht möglich): Nun ist x ein alias für a[0], wir
schreiben x für a[0]. Es wird in p also a[0] =a[0]+2; ausgeführt und 12 und 20 ausgegeben.

Parameterübergabe und Laufzeitstapel

- Speicherbild beim Ablauf einer einfachen Funktion in Java, C/C++



Benötigte Verwaltungsgrößen

Statischer Verweis (static link) auf den Rahmen, in dem globale Variablen gespeichert sind

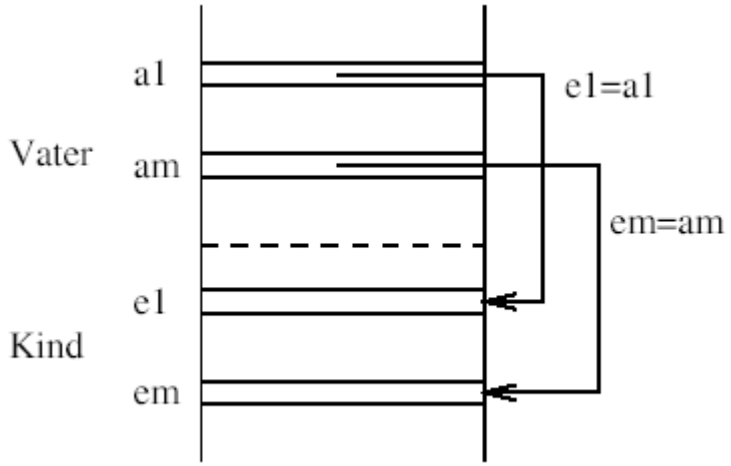
Dynamischer Verweis (dynamic link) auf den Rahmen der Prozedur-Inkarnation, aus der der gegenwärtige Prozeduraufruf getätigt wurde. Das ist immer der Rahmen direkt unterhalb auf dem Stapel.

Die **Rücksprungadresse** (return address) spezifiziert die Anweisung, mit der die Berechnung nach Beendigung des Unterprogramm-Aufrufs fortfährt.

Parameterübergabe und Laufzeitstapel: Werteaufruf

Der Einfachheit halber seinen die aktuellen Parameter Variablen im Block des Vaters. Die formalen Parameter sind lokale Variablen im Block des Sohnes. Bei der **Werteübergabe** werden die Werte der aktuellen Parameter (Ausdrücke oder Variablen) vom Vater zum Sohn kopiert. Dies geschieht in einem vom Übersetzer automatisch erzeugten Zuweisungsblock zu Beginn der Sohn-Prozedur.

```
Te1 e1 = a1;
...
Tem em = am;
```



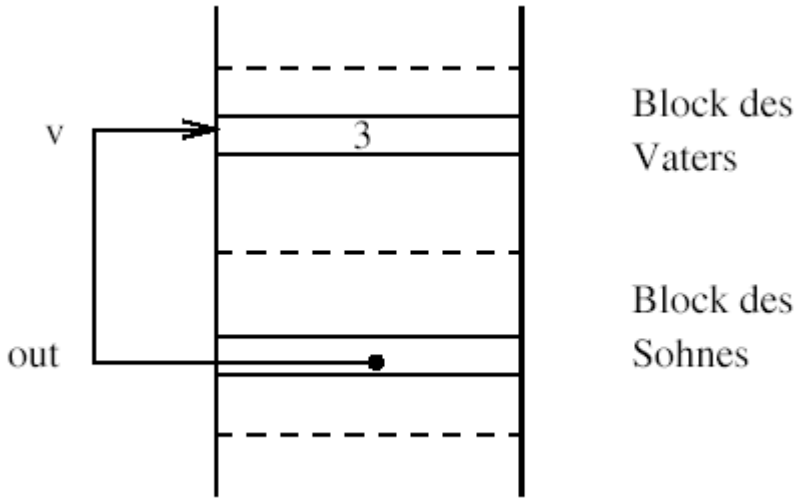
Parameterübergabe und Laufzeitstapel: Werteaufruf

- Nach Beendigung des Unterprogramms werden die lokalen Variablen aufgelöst und der Speicherrahmen freigegeben
- Die Werte der formalen Parameter werden **nicht** an die aktuellen Parameter zurück überwiesen, falls Werteübergabe vereinbart war
  - Dieser Fall gilt grundsätzlich für C und Java
- Diese Übergabetechnik kann im Prinzip aber ebenso in umgekehrter Richtung auf Ausgabeparameter angewendet werden
  - Im früher für FORTRAN verwendeten Mechanismus der **Werte- und Resultatsübergabe** (call by value and result) werden beim Austritt aus der Prozedur die Werte der formalen Ausgabeparameter an die aktuellen Ausgabeparameter zugewiesen

Parameterübergabe und Laufzeitstapel: Referenzaufruf

- Grundidee der **Referenzübergabe** ist es, nicht den Wert selbst zum Kind zu kopieren, sondern nur eine Referenz auf das Objekt zu übergeben, die immer gleich groß ist (z. B. ein 32-Bit-Zeiger)
- Das Kind arbeitet effektiv also mit einer Referenz-variablen vv als formalem Parameter, die eine Referenz-stufe höher ist als der Typ des aktuellen Parameters v

- Sei out ein formaler und v ein aktueller Ausgabeparameter.



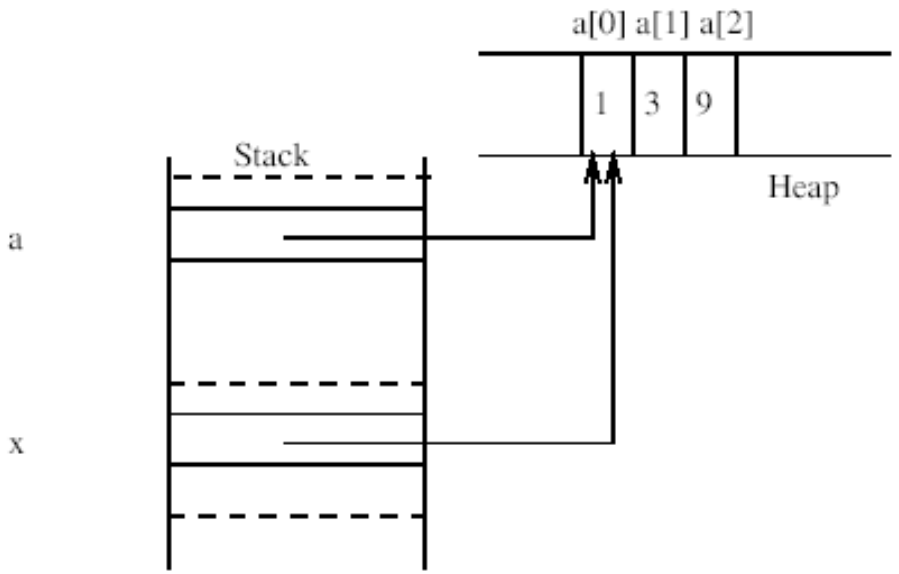
- **Bemerkung:** In C, wo es keine Referenzübergabe gibt, kann man den Effekt dieses Mechanismus dadurch erhalten, dass man in der Kind-Prozedur Zeiger-Variablen benutzt und die zusätzliche Indirektion beim Zugriff explizit ausprogrammiert
  - In C++ existieren dagegen allgemeine Referenzvariablen

Parameterübergabe und Laufzeitstapel

- **Beipiel:** Werteübergabe bei Referenzvariablen

```
void father() {
  int[] a = new int[3];
  a[0]=1; a[1]=3; a[2]=9;
  a=son(a);
  // Punkt 4
}

int[] son(int[] x) { // Punkt 1
  x[0]=7; // Punkt 2
  x = new int[2]; // Punkt 3
  return x;
}
```

Speicherbild der eingeschränkten Referenzübergabe in Java am Punkt 1. Das Speicherbild sieht nach der Übergabe eines Parameters vom Klassentyp int[] wie folgt aus:

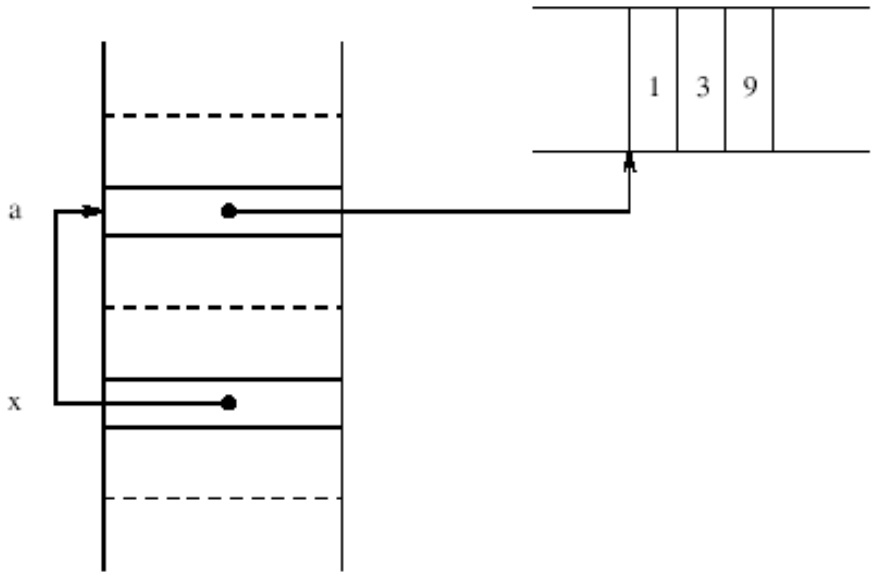


Parameterübergabe und Laufzeitstapel

- **Beipiel bei Referenzübergabe**

```
void father() {
  int[] a = new int[3];
  a[0]=1; a[1]=3; a[2]=9;
  a=son(a);
  // Punkt 4
}

int[] son(int[] x) { // Punkt 1
  x[0]=7; // Punkt 2
  x = new int[2]; // Punkt 3
  return x;
}
```

Speicherbild bei uneingeschränkter Referenzübergabe des Parameters x am Punkt 1 (in C und Java nicht möglich).
Wir hätten eine weitere Indirektion erhalten:

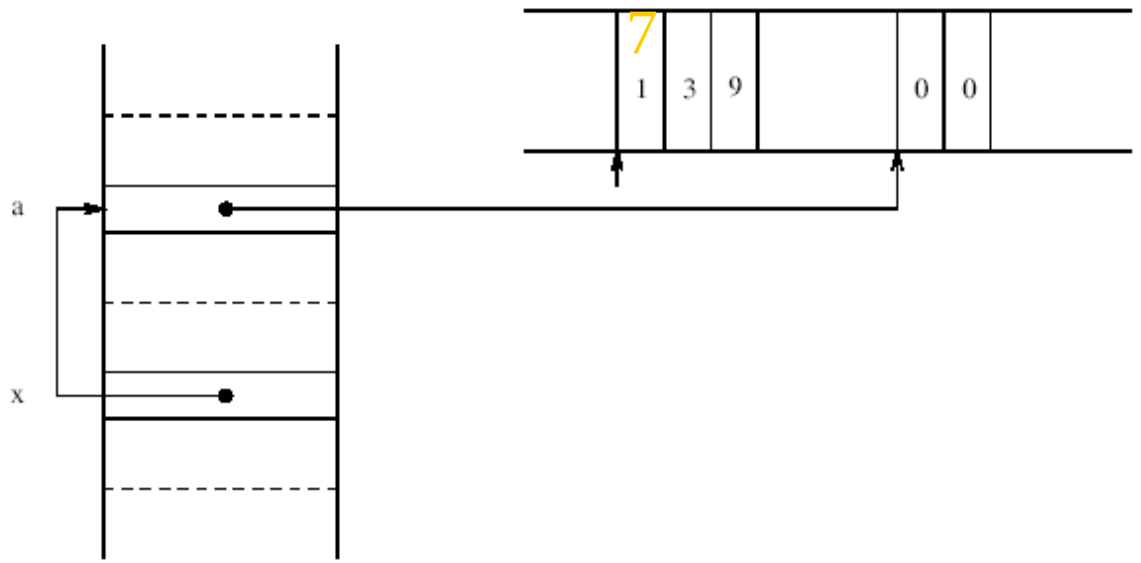


Parameterübergabe und Laufzeitstapel

- **Beipiel bei Referenzübergabe**

```
void father() {
  int[] a = new int[3];
  a[0]=1; a[1]=3; a[2]=9;
  a=son(a);
  // Punkt 4
}

int[] son(int[] x) { // Punkt 1
  x[0]=7; // Punkt 2
  x = new int[2]; // Punkt 3
  return x;
}
```

Speicherbild bei uneingeschränkter Referenzübergabe des Parameters x am Punkt 3 (in Java nicht möglich).
Änderung des Wertes auch beim Vater

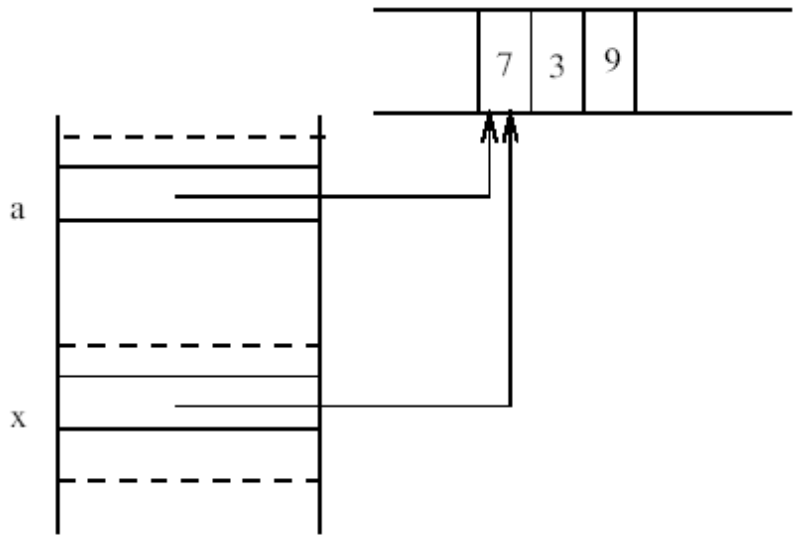


Parameterübergabe und Laufzeitstapel

- **Beipiel (Forts.):** Werteübergabe bei Referenzvariablen

```
void father() {
  int[] a = new int[3];
  a[0]=1; a[1]=3; a[2]=9;
  a=son(a);
  // Punkt 4
}

int[] son(int[] x) { // Punkt 1
  x[0]=7; // Punkt 2
  x = new int[2]; // Punkt 3
  return x;
}
```

Speicherbild der eingeschränkten Referenzübergabe in Java am Punkt 2. Die Zuweisung an ein Element ändert den Wert des Arrays auch beim Vater.

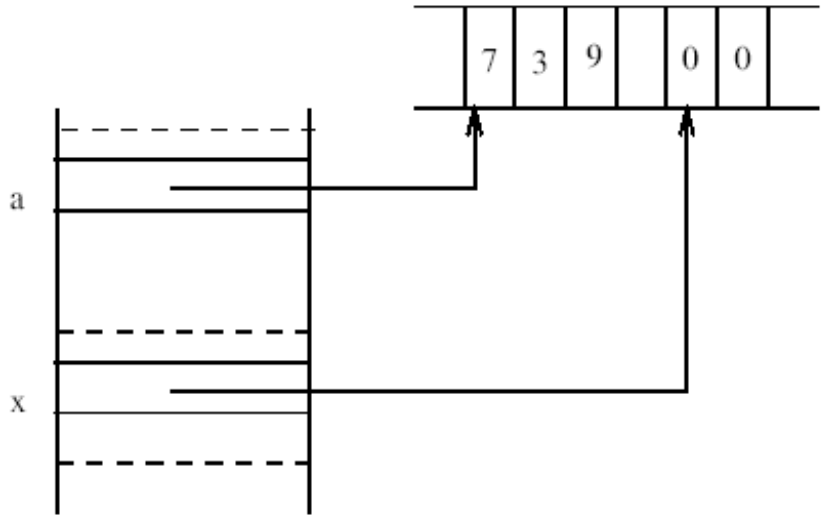


Parameterübergabe und Laufzeitstapel

- **Beipiel (Forts.):** Werteübergabe bei Referenzvariablen

```
void father() {
  int[] a = new int[3];
  a[0]=1; a[1]=3; a[2]=9;
  a=son(a);
  // Punkt 4
}

int[] son(int[] x) { // Punkt 1
  x[0]=7; // Punkt 2
  x = new int[2]; // Punkt 3
  return x;
}
```

Speicherbild der eingeschränkten Referenzübergabe in Java am Punkt 3. Das new ändert nur die Referenz im Sohn.

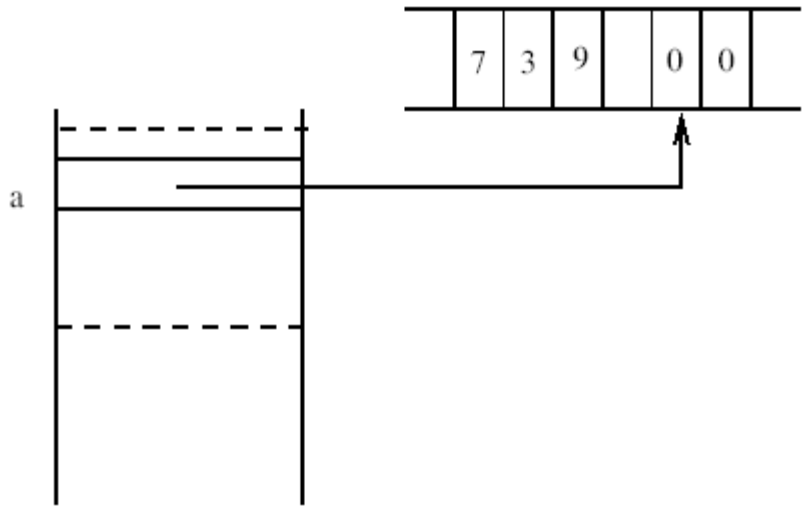


Parameterübergabe und Laufzeitstapel

- **Beipiel (Forts.):** Werteübergabe bei Referenzvariablen

```
void father() {
  int[] a = new int[3];
  a[0]=1; a[1]=3; a[2]=9;
  a=son(a);
  // Punkt 4
}

int[] son(int[] x) { // Punkt 1
  x[0]=7; // Punkt 2
  x = new int[2]; // Punkt 3
  return x;
}
```

Speicherbild der eingeschränkten Referenzübergabe in Java am Punkt 4 aufgrund der Zuweisung.



Parameterübergabe und Laufzeitstapel

- Ein mögliches Speicherbild in C
  - Reihungen oder andere komplexe Objekte können auch auf dem Stack angelegt werden
    - ⇨ Sehr gefährlich (etwa dangling pointers)