Pseudocode

(Stand: 15.03.2022)

Wir verwenden folgende Konventionen in unserem Pseudocode.

Beispiel

```
1   for j ← 1 to dim(A)-1 do
2       for k ← dim(A) downto j+1 do
3           // Kleinere Elemente wandern nach vorne
4           if A[k-1].key > A[k].key then
5              vertausche A[k-1] und A[k]
6           end if
7       end for
8       // Gib den Zwischenstand nach einem Durchlauf aus
9       ausgeben A
10  end for
```

Anweisungsblöcke

Anweisungsblöcke werden im Folgenden mit dem Nicht-Terminalsymbol `<Anweisungen>` bezeichnet. Anweisungsblöcke bestehen aus einer durch Schlüsselworte (unten in Fettdruck) eingegrenzten Folge von einer und mehr Einzelanweisungen.

if <Bedingung> **then** <Anweisungen> **end if**

if <Bedingung> **then** <Anweisungen> **else** <Anweisungen> **end if**

for <Schleifenkontrolle> **do** <Anweisungen> **end for**

while <Bedingung> **do** <Anweisungen> **end while**

repeat <Anweisungen> **until** <Bedingung>

Die einzelnen Anweisungen eines Blocks beginnen jeweils in einer eigenen Zeile gleicher Einrücktiefe. In obigem Beispiel beginnt die erste **for**-Schleife in Zeile 1 und der zu dieser **for**-Schleife gehörende Anweisungsblock wird eingegrenzt durch die Schlüsselworte **do** (Zeile 1) und **end for** (Zeile 10). Der Anweisungsblock innerhalb der **for**-Schleife besteht aus der **for** -Anweisung (Zeilen 2-7) und der Anweisung `ausgeben A` (Zeile 12).

Anweisung

Eine *Anweisung* kann sein

- ein Satz wie „tausche A[k-1] und A[k]“, „Schiebe alle Elemente ab `a[i]` um eine Position nach rechts“ oder „Füge Element `x` an Position `i` ein“
- eine Zuweisung (s.u.) wie `y ← (x+z)*2` oder `i ← j ← e`
- eine Kontrollstruktur wie **if - then - else - end if**
- ein Methodenaufruf (s.u.)
- `return` (s.u.)

// ist ein Zeilenkommentar

Die //-Zeichen zeigen wie in Java an, dass der Rest der Zeile ein Kommentar ist.

Operatoren

Im Pseudocode werden als Operatoren die bekannten Java-Operatoren benutzt. So wird z.B. der Gleichheitsoperator durch ein doppeltes Gleichheitszeichen dargestellt. Abweichend davon werden Zuweisungen durch ein ← dargestellt. Anstelle von **&**, | und **!** werden **and**, **or** und **not** für die Booleschen Grundoperatoren verwendet.

Mehrfachzuweisungen i←j←e

Eine Mehrfachzuweisung der Form `i ← j ← e` weist den beiden Variablen `i` und `j` den Wert des Ausdruck `e` zu. Die Mehrfachanweisung ist äquivalent zu den Einzelanweisungen `j ← e` gefolgt von `i ← j` und entspricht der Java-Anweisung `i = j = e`.

Die Semantik der if - then (- else)-Anweisung

- Bei der **`if-then`**-Anweisung wird zunächst die Bedingung geprüft. Nur wenn sie erfüllt ist, werden die Anweisungen zwischen **`then`** und **`end if`** ausgeführt.
- Bei der **`if-then-else`**-Anweisung werden die Anweisungen zwischen **`then`** und **`else`** ausgeführt, wenn die Bedingung erfüllt ist. Wenn sie nicht erfüllt ist, werden die Anweisungen zwischen **`else`** und **`end if`** ausgeführt.

```
if <Bedingung> then
    <Anweisungen>
end if

if <Bedingung> then
    <Anweisungen>
else
    <Anweisungen>
end if
```

Die Semantik der while-Schleife

- Bei der **`while`**-Schleife wird zunächst die Bedingung geprüft.
- Ist sie erfüllt, werden die Anweisungen in der Schleife ausgeführt.
- Nach Ausführung der Anweisungen wird wiederum die Bedingung geprüft.
- Ist sie erfüllt, werden die Anweisungen wiederholt und so fort.
- Wenn die Bedingung nicht erfüllt ist, werden die Anweisungen nicht (mehr) ausgeführt. Ist die Bedingung von vornherein nicht erfüllt, werden die Anweisungen in der Schleife also auch nicht ein einziges Mal ausgeführt (kopfgesteuerte Schleife).

```
while <Bedingung> do
    <Anweisungen>
end while
```

Die Semantik der repeat-Schleife

- Bei der `repeat`-Schleife werden zuerst die Anweisungen zwischen `repeat` und `until` ausgeführt.
- Danach wird die angegebene Bedingung geprüft.
- Wenn sie erfüllt ist, wird die Schleife beendet, ansonsten werden die Anweisungen wiederholt und anschließend wird erneut geprüft.
- Zu beachten ist, dass bei dieser Schleife die Anweisungen in jedem Fall mindestens einmal ausgeführt werden (fußgesteuerte Schleife).

```
repeat
    <Anweisungen>
until <Bedingung>
```

Die Semantik der for-Schleife

- Bei der **`for`**-Schleife wird zur Schleifenkontrolle zunächst ein Zähler auf einen bestimmten Startwert gesetzt.
- Als Vorbedingung der Schleife wird geprüft, ob der Zähler einen bestimmten Endwert überschritten hat.
- Wenn dies nicht der Fall ist, werden die zu wiederholenden Anweisungen ausgeführt und anschließend der Zähler um die Schrittweite 1 erhöht (oder bei **`downto`** um 1 erniedrigt). Ist eine andere Schrittweite erforderlich, wird diese durch **`step`** `<Schrittweite>` explizit angegeben.
- Danach erfolgt wieder die Prüfung.

```
for <Zähler> ← <Startwert> to <Endwert> do
    <Anweisungen>
end for

for <Zähler> ← <Startwert> to <Endwert> step <Schrittweite> do
    <Anweisungen>
end for

for <Zähler> ← <Startwert> downto <Endwert> do
    <Anweisungen>
end for

for <Zähler> ← <Startwert> downto <Endwert> step <Schrittweite> do
    <Anweisungen>
end for
```

Prozeduren und Funktionen

Prozeduren und Funktionen werden durch Angabe des Namens und Aufzählung der Parameter in Klammern, gefolgt von den Anweisungen, deklariert:

```
potenziere(a, b)
    <Anweisungen>
```

Bei Funktionen muss es eine Zeile **`return`** `wert` geben. Der Aufruf erfolgt durch Angabe des Prozedurnamens und der aktuellen Parameter wie bei `potenziere(1, 3)`.

Variablen sind lokal bezüglich der gegebenen Methode

Variablen sind lokale Variablen. Es werden keine globalen Variablen verwendet ohne dies explizit anzugeben.

Die Parameterübergabe erfolgt call-by-value (analog zu Java)

Die aufgerufene Methode erhält eine Kopie der Parameter. Wertzuweisungen werden von der aufrufenden Methode nicht gesehen. Bei Objekten wird die Referenz auf die Daten kopiert, nicht aber die Objekte selbst. Beispiel: Wenn `x` der Parameter der aufgerufenen Methode ist, dann ist die Wertzuweisung `x ← y` nicht in der aufrufenden Methode sichtbar, aber die Zuweisung `x.a ← 3` ist sichtbar.

Ein- und mehrdimensionale Arrays

Bei eindimensionalen Arrays greift man mit dem Namen des Arrays gefolgt von einem Index auf Array-Elemente zu: `A[i]` ist das `i`-te Element des Arrays `A`.

Die Notation `..` wird benutzt, um einen Wertebereich innerhalb eines Arrays zu beschreiben: `A[1..n]` besteht aus den Elementen `A[1]`, `A[2]` bis `A[n]`; die Arraygrenzen verlaufen von 1 bis zur Länge des Arrays, die mit `dim(A)` ermittelt wird.

Auf die Elemente mehrdimensionaler Felder wird durch den Namen des Feldes, gefolgt von eckigen Klammern zugegriffen, in denen durch Komma getrennt die Positionsangaben stehen: `Matrix[x, y]` beschreibt das Array-Element, das in der `x`-ten Zeile und `y`-ten Spalte steht.

Für die Dimensionsangaben wird die Notation `..` benutzt: `M[1..m, 1..n]` beschreibt eine zweidimensionale `m×n`-Matrix. Die Länge des Spaltenvektors (Anzahl der Zeilen), also die Größe der ersten Dimension, kann mit `dim(M, 1)` und die Länge eines Zeilenvektors (Anzahl der Spalten) mit `dim(M, 2)` ermittelt werden.

b.attr beschreibt das Attribut attr eines Objektes b

Zusammengesetzte Daten werden typischerweise durch Attribute in Objekten dargestellt. Auf ein bestimmtes Attribut greift man mit dem Objektnamen gefolgt von einem Punkt und dem Attributnamen zu. `b.attr` ist das Attribut `attr` des Objektes `b`.

b.meth() beschreibt den Aufruf einer Objektmethode meth()

Objektmethoden werden durch den Objektnamen gefolgt von einem Punkt und dem Methodennamen aufgerufen. `b.meth()` ist der Aufruf der Methode `meth()` des Objektes `b`.

Array- oder Objekt-Variablen sind Referenzen

Eine Variable, die ein Array oder ein Objekt repräsentiert, wird wie eine Referenz auf das Array oder Objekt behandelt. Für alle Attribute `a` eines Objektes `x` hat `y ← x` auch `y.a ← x.a` zur Folge. Wenn anschließend `x.a ← 3` ist nicht nur `x.a==3`, sondern auch `y.a`. Mit anderen Worten, `x` und `y` zeigen nach der Zuweisung `y ← x` auf ein und dasselbe Objekt.

Wenn eine Referenz noch nicht auf ein Objekt zeigt, erhält sie den Wert **`nil`**.