5. Aufzählungstypen

Schema:

type <Bezeichner> is (<Auflistung>)

Beispiele:

```
type Freunde is ( Daniel, Vera, Anke,
                  Thomas, Peter, Melanie );

type Wochentage is ( Mo, Di, Mi, Do, Fr, Sa, So );

type Studienfächer is ( Informatik, Elektrotechnik,
     Mathematik, Softwaretechnik, Wirtschaftsinformatik,
     Computerlinguistik, Autip, BWL, TP);

type Farbe is (weiß, schwarz, rot, blau, gelb, grün, lila, orange);

type Helligkeit is (hellgrau, grau, dunkelgrau, weiß, schwarz);
```

Variablendeklaration wie üblich:

```
X : Wochentage; Y : Farbe; Z : Helligkeit;
```

Wertzuweisung:

```
X := Fr;   Y := schwarz;   Z := schwarz;
```

Einschränkung:

Wenn ein Bezeichner in einer Auflistung verwendet wird, darf er nicht zugleich als Variable benutzt werden (er darf aber in verschiedenen Aufzählungstypen auftreten, siehe "schwarz" oben).

Nach obigen Deklarationen darf also nicht

Mo: Integer; oder blau: Farbe;

folgen.

Ada besitzt strenge Typisierung. Die Zuweisung

X := <Ausdruck>

ist nur zulässig, wenn X und <Ausdruck> genau den gleichen Datentyp besitzen.

Daher gibt es keine Missverständnisse bei:

Y := schwarz; und Z := schwarz;

Y := schwarz; gehört zum Typ "Farbe"

Z := schwarz; gehört zum Typ "Helligkeit"

Manchmal muss man jedoch angeben, welches Objekt gemeint ist. Dann schreibt man

Farbe'(schwarz) bzw. Helligkeit'(schwarz)

In Ada ist jeder Aufzählungstyp angeordnet in der aufgelisteten Reihenfolge.
z.B.: Mo < Di < Mi < Do < Fr < Sa < So.

Es gibt nun für Aufzählungstypen folgende "Attribute":

First	erstes Element des Datentyps
Last	letztes Element des Datentyps
Succ	nächstes Element des Datentyps (sofern existiert)
Pred	vorheriges Element des Datentyps (" ")
Pos	Nummer des Elements in der Aufzählung
Val	Element, das zu dieser Nummer gehört

Die Nummerierung beginnt in Ada stets mit der Null !!

Beispiele und Schreibweisen:

Farbe'First (= weiß) Freunde'Last (= Melanie)
Studienfächer'Succ(Antip) (= BWL)
Wochentage'Pos(Fr) (= 4)
Farbe'Val(3) (= blau)
X := Sa; X := Wochentage'Val(Wochentage'Pos(X) - 3);
(X hat anschließend den Wert Mi)

6. Unterbereiche

Durch Einschränkungen gewinnt man aus einem Datentyp einen "Unter-Datentyp" (subtype). Der Unterdatentyp übernimmt alle Operationen, die auf dem Datentyp erlaubt sind. Sie werden ausgeführt, als wäre man im (Ober-) Datentyp und am Ende wird geprüft, ob man im Unter-Datentyp geblieben ist.

Die "Einschränkung" ist von der Form

range <einf. Ausdruck> .. <einf. Ausdruck>

Standardbeispiel: Datumsangaben

```
subtype Monat is Integer range 1 .. 12;
subtype Tag is Integer range 1 .. 31;
subtype Februartage is Tag range 1 .. 29;
subtype Arbeitstage is Wochentage range Mo .. Fr;
subtype Heimarbeit is Arbeitstage range Di .. Do;
```

Unterdatentypen werden "durchnummeriert":

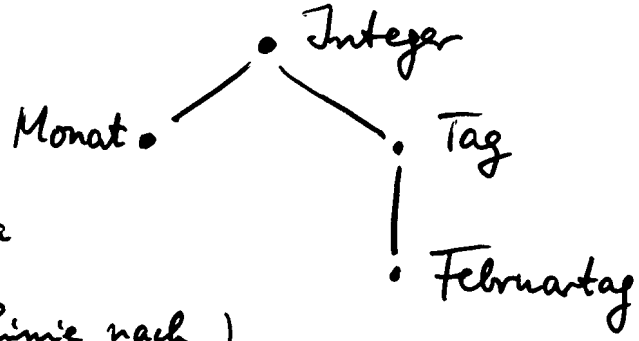
Tag ist 1. Unterdatentyp von Integer,
Februartage ist 2. " " " ,
Arbeitstage ist 1. " " Wochentage,
Heimarbeit ist 2. " " " .

Eine Variable wird als Variable des "Oberdatentyps" aufgefasst, aber in ihr dürfen nur Werte gespeichert werden, die die Einschränkung des Unterdatentyps erfüllen und von diesem Typ sind.

Beispiele

A : Integer; B : Tag; C : Monat; D : Februartage;

A := 40;
B := A / 4;
C := A / 4;
B := C ist verboten, da anderer Datentyp (nicht in gerader Linie nach oben!)



Mittels <Datentyp>(...)

kann man zum Ober- oder Unterdatentyp übergehen, also

B := Integer (C); (ist erlaubt, sofern die Einschränkung erfüllt ist)
C := Integer (B); (")

B := 7 * B / 8 + 1; (")

7. Felder

Zusammenfassung von Variablen des gleichen Datentyps.
Zugriff über einen Index.
Deklarations-Schema

type <Bezeichner> is

array (____) of <Datentyp>;

Liste von "Indextypen" [deren Anzahl ist die Dimension des Feldes] ↑ "Komponententyp"

Beim "Indextyp" gibt es zwei Möglichkeiten:

spezifiziert (constrained) und unspezifiziert (unconstrained)

Spezifizierter Fall:

Die Indexdatentypen sind endliche Bereiche, deren Größe bei der Deklaration ausgerechnet werden kann.
Der Typ "Float" (also reelle Zahlen) darf nicht im Indexdatentyp verwendet werden, d.h. als Index sind nur "diskrete" Typen erlaubt.
Es ist zulässig, dass die Größe der Indextypen erst zur Laufzeit bekannt sind und nicht bereits zur Übersetzungszeit (dynamisches Feld im Gegensatz zum statischen Feld). Man muss dann aber Blöcke verwenden (begin ... end), siehe später.

Beispiele:

Name des Datentyps

statisches Feld

```
type Stundenplan is
    array ( Arbeitstage ,
            Integer range 8..19)
    of Studienfächer ;
```

1. Indextyp (Aufzählungs-Untertyp)

2. Indextyp (Untertyp von ℤ)

Komponentyp (hier ein Aufzählungstyp)

dynamische Felder

```
type Matrix is array (1..N, 1..M) of Float;
type Code is array ( Character ) of Integer range 1..N;
type Speicher is array (1..2**N) of Boolean ;
```

Unspezifizierter Fall:

Will man sich auf die Größe eines Indextyps nicht bereits am Anfang festlegen, so schreibt man

range <> (<> wird "box" gesprochen)

und legt die konkrete Bereichsbeschränkung erst fest, wenn man sie im Programm braucht. Man kann den unspezifizierten Fall für Prozeduren und Funktionen und als Platzhalter für künftige Konkretisierungen verwenden.

type Vektor is array (Integer range <>) of Float ;

Später konkretisiert man dies zu

X : Vektor (-20 .. 20) ;
Y : Vektor (1 .. N * N) ;

Ein Unter-Programm oder Operatoren, die für "Vektor" formuliert sind, können dann unmittelbar für X bzw. Y übernommen werden.

Feld-Attribute

First (i)	erster Element des i-ten Indextyps
Last (i)	letztes " " " "
Length (i)	Anzahl der Elemente des i-ten Indextyps
Range (i)	i-ter Indextyp

Diese werden (abgetrennt durch Apostroph) an den Variablennamen angehängt. Beispiel: Matrixmultiplikation:

```
type Matrix is array (1..N, 1..N) of Float; ...
A, B, C : Matrix; ...
for i in A'Range(1) loop
for j in B'Range(2) loop
    C(i,j) := 0.0;
    for k in A'Range(2) loop
        C(i,j) := C(i,j) + A(i,k) * B(k,j) end loop;
end loop; end loop;
```

(Hier hätte man auch eine unspezifizierte Matrix verwenden können.)