Aufzählungs- und Unterbereichstypen

- Aufzählungstyp ("enumeration") (cont'd)
  - Beispiele:
    - mit Typdefinition

```
TYPE
  Wochentag = (Montag, Dienstag, Mittwoch, Donnerstag,
               Freitag, Samstag, Sonntag);
  FarbPalette = (Weiss, Rot, Blau, Gelb, Gruen, Schwarz);

VAR
  Tag        : Wochentag;
  Farbe      : FarbPalette;
```

    - wäre BOOLEAN nicht schon Standard

```
TYPE
  BOOLEAN = (FALSE, TRUE)
```

Aufzählungs- und Unterbereichstypen

- Aufzählungstyp ("enumeration") (cont'd)
  - Bemerkungen:
    - Elemente müssen eindeutig definiert sein, dürfen also nicht in 2 Typen vorkommen
    - Bezeichner dürfen nur in **einer** festgelegten Bedeutung vorkommen
      Beispiele:
```
Karte : (7, 8, 9, Bube, Dame, König, 10, As)
Tag   : (MO, DI, MI, DO, FR, SA, SO)
aber erlaubt:
Tag   : (Mo, Di, Mi, Do, Fr, Sa, So)
```
    - die Elemente einer Aufzählung erhalten analog ihrer Position in der Aufzählung eine Ordnungszahl zugeordnet
  - Operationen und Aktionen
    - Wertzuweisung wie gewohnt: `Tag := Dienstag; Farbe := Blau;`
    - durch die Def. der Ordnung sind Vergleiche möglich {=,<>,<,<=,>,>=}
    - Funktionen
      - Minimum, Maximum: MIN(T) = a_0 , MAX(T) = a_n ← Typ T = (a_0,a_1,...,a_n)
      - Ordnungszahl- und Wertfunktion: ORD(...), VAL(...)
        mit: ORD(x) = VAL(CARDINAL,x)
      - Nachfolger und Vorgänger: INC(...), DEC(...)

Aufzählungs- und Unterbereichstypen

➢ Aufzählungstyp (“enumeration”) (cont'd)

- Bsp: MIN, MAX, ORD, INC, DEC bei einem Aufzählungstyp

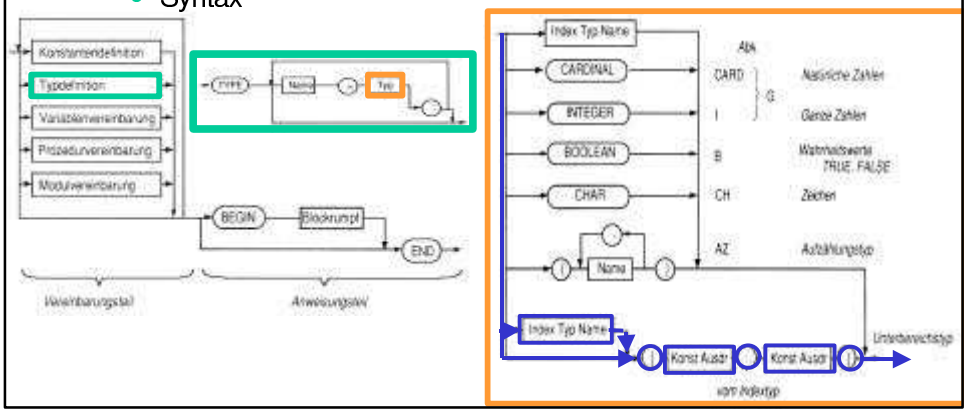
```
TYPE
  Spielkarte : (Sieben, Acht, Neun, Bube, Dame,
                Koenig, Zehn, As);
VAR
  Karte : Spielkarte;

BEGIN
  ...
  WriteCard(ORD(MIN(Spielkarte)), 2);
  WriteCard(ORD(MAX(Spielkarte)), 2);

  Karte := Neun;
  WriteCard(ORD(Karte), 2);

  INC(Karte);
  WriteCard(ORD(Karte), 2); WriteLn;

  DEC(Karte, 2);
  WriteCard(ORD(Karte), 2); WriteLn;

END
```

= ORD(Sieben) = 1
= ORD(As) = 8
= ORD(Neun) = 3
Karte = Bube
= ORD(Bube) = 4
Karte = Acht
= ORD(Acht) = 2

Aufzählungs- und Unterbereichstypen

➢ Unterbereichstyp (“subrange”)

- Festlegung eines Unterbereichs des Definitionsbereichs eines ...
  - vordefinierten Typs
  - eigens definierten (Ordinal-)Typs
- der Grundtyp ist der “Träger (host type)”
- Syntax



Aufzählungs- und Unterbereichstypen

- Unterbereichstyp ("subrange") (cont'd)
  - Beispiele:
    - Typdefinition

```
TYPE
  Buchstabe  = ["a" .. "z"];
  Ziffer     = [0 .. 9];          (*Grundtyp CARDINAL*)
  ZahlInt    = INTEGER[1 .. 10];
  Wochentag  = (Mo, Di, Mi, Do, Fr, Sa, So);
  VL_Tag     = [Mo .. Fr];
  BYTE       = [0 .. 255];
```

    - Variablendefinition

```
VAR
  CAPITAL    : ["A" .. "Z"];
  OktZiffer  : Ziffer[0 .. 7];
```

Aufzählungs- und Unterbereichstypen

- Unterbereichstyp ("subrange") (cont'd)
  - Hinweise:
    - Verwendung eines AufzTypName (hier INTEGER, Ziffer) zur Verdeutlichung des Grundtyps (Bsp.: CARDINAL vs. INTEGER)
    - der Trägerdatentyp bestimmt die Menge der erlaubten Operationen
    - bei Operationen auf Unterbereichstypen dürfen die Ergebnisse den Wertebereich nicht verlassen
    - Unterbereiche auf REAL nicht erlaubt

Funktion VAL(T,x)

Vereinbarungen:

```
TYPE
    Tage          = (So, Mo, Di, Mi, Do, Fr, Sa);
    Wochentage    = [Mo .. Fr];

VAR
   i, j, z : INTEGER;
   r       : REAL
```

Wertzuweisungen:

```
i := 42; j := -1; z := 0; r := -2.7;
```

VAL liefert folgende Ergebnisse:

```
VAL(CARDINAL, i)    = 42
VAL(CARDINAL, j)    = Fehler
VAL(CARDINAL, r)    = Fehler
VAL(INTEGER, i)     = 42
VAL(INTEGER, r)     = -2
VAL(REAL, i)        = 42.0
VAL(REAL, TRUE)     = Fehler
VAL(LONGREAL, r)    = -2.7
VAL(CHAR, z)        = 0C
VAL(CHAR, r)        = Fehler
VAL(BOOLEAN, 0)     = FALSE
VAL(Tage, 5)        = Fr
VAL(Wochentage, 5)  = Fr
VAL(Wochentage, z)  = Fehler
```

Beziehungen mit VAL

```
CHR(x)     = VAL(CHAR, x)
ORD(x)     = VAL(CARDINAL, x)
INT(x)     = VAL(INTEGER, x)
TRUNC(x)   = VAL(CARDINAL, x)
FLOAT(x)   = VAL(REAL, x)
LFLOAT(x)  = VAL(LONGREAL, x)

x vom Typ T:
INC(x,n) ↔ x:=VAL(T,VAL(INTEGER,x)+n)
```

Aufzählungs- und Unterbereichstypen

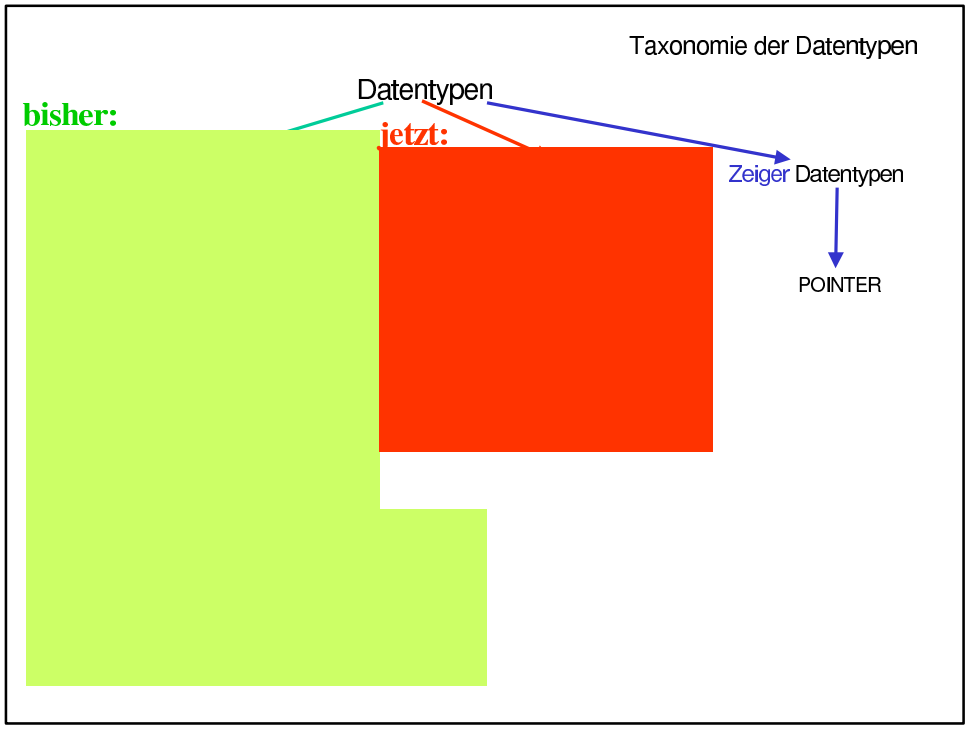
➢ Kompatibilität (auch für strukturierte Daten)

- Typkompatibilität
  strenge Regeln bzgl. der verträglichkeit von verschiedenen Typen in Ausdrücen und Zuweisungen!
  - 2 Typen sind identisch, wenn sie in einerTypdefinition gleichgesetzt werden
  - 2 Typen haben den gleichen Grundtyp, wenn sie Unterbereiche des gleichen Typs sind.
- Ausdruckskompatibilität
  Alle Operanden in einem Ausdruck haben den gleichen Typ oder den gleichen Grundtyp
- Zuweisungskompatibilität
  Variable W sei vom Typ V, Ausdruck A von Typ E;
  Zuweisung W := A ist möglich, falls (Auswahl):
  - V und E sind gleich oder haben gleichen Grundtyp
  - V hat Grundtyp CARDINAL,E hat Grundtyp INTEGER oder umgekehrt
  - V hat Grundtyp CHAR und E ist einelem. oder leere Stringkonstante
  - V ist vom Typ ADDRESS,E ist vom Typ POINTER oder umgekehrt

Strukturierte Daten

Datenstrukturen

- Einordnung
  - bisher: „nur“ einfache Objekte eines Typs → unstrukturierte Typen
  - jetzt:
    - strukturierte Objekte (statisch) → Komponenten
    - dynamische Objekte → Zeiger, Dateien



Strukturierte Daten

Datenstrukturen

- Unterscheidungsmerkmale
  - Typ der Komponenten
    - alle Komponenten vom selben Typ (homogene Struktur)
    - Komponenten sind i.A. unterschiedlichen Typs (heterogene Struktur)
    - übrigens: Komponenten können selbst strukturierte Objekte sein
  - Anzahl der Komponenten
    - fest vorgegeben, schon zur Übersetzungszeit bekannt, bleibt während der Programmlaufzeit konstant
    - nicht fest vorgegeben; wird während der Programmlaufzeit einmal bestimmt, bleibt für den Rest der Programmlaufzeit konstant
    - variabel, kann sich während der Programmlaufzeit beliebig ändern (dynamische Liste)
  - Zugriff auf die Komponenten
    - sequentiell, d.h. nur in bestimmter Reihenfolge (→ Dateien)
    - direkt über Index (Zahl, Nummer) oder Komponenten-Bezeichner (Name)
    - Zugehörigkeitstest (m∈M? → true/false); Komponenten nicht direkt ansprechbar

ARRAYs (Felder)

- Allgemeines
  - Verwendung
    Gruppierungsmöglichkeit (Struktur!) einer fest definierten Anzahl von Variablen derselben Charakteristik (=(Daten-)Typ)
  - Deklaration

```
VAR
  feld  : ARRAY[Typ 1] OF Typ 2;
```

    - Typ 1: Indextyp, muss Ordnungstyp sein
    - Typ 2: jeder beliebige, auch selbstdefinierte Typ
    - `a  :  ARRAY[1..N],[1..N] OF REAL`
      ist eigentlich nur eine Abkürzung für:

```
a  :  ARRAY[1..N] OF
        ARRAY[1..N] OF REAL
```

      Einrückung zeigt Hierarchie
  - Komponenten von Feldern können selbst wieder strukturiert sein, also z.B. Felder

ARRAYs (Felder)

➢ Allgemeines (cont'd)

- Beispiele zu den Indextypen

```
TYPE
  TAGE       =  CARDINAL[1 .. 7];

VAR
  tabelle    :  ARRAY[CHAR] OF CHAR;
  speicher   :  ARRAY[0 .. 1000] OF INTEGER;
  freierTag  :  ARRAY[TAGE] OF BOOLEAN;
  matrix     :  ARRAY[1 .. 3],[1 .. 5] OF REAL;
```

Übersetzung eines Zeichensatzes in einen anderen

Ordinaltyp

lineares Speichersegment

Ordinaltyp
Unterbereich aus
CARDINAL/INTEGER

arbeitsfreie Wochentage

Ordinaltyp
oben definierter Typ
Unterbereich aus CARDINAL

3 x 5 Matrix

Ordinaltyp
Unterbereich aus
CARDINAL/INTEGER

ARRAYs (Felder)

➢ Allgemeines (cont'd)

- Beispiele zu gültigen Vereinbarungen von ARRAYs

```
CONST
    n = 8;
TYPE
    Wort     = ARRAY[0..3] OF CHAR;
    Vektor   = ARRAY[1..n] OF REAL;
    Letter   = ARRAY["a".."z"] OF INTEGER;
    Farbe    = (rot, gelb, blau);
    Fahne    = ARRAY[1..3] OF Farbe;
    Muster   = ARRAY Farbe OF BOOLEAN;
    Palettel = ARRAY[1..n] OF ARRAY Fa    OF BOOLEAN;
    Palette2 = ARRAY[1..n], Farbe    BOOLEAN;
    Matrix   = ARRAY[1..n],[1..n] OF REAL;
    Matrixl  = ARRAY[1..n] OF Vektor;

VAR
    name,vorname     : Wort;
    feld             : Matrix;
    feld1            : Matrixl;
    klBuchst         : Letter;
    muster           : Muster;
    identity         : Matrix;
    v                : Vektor;
    z                : ARRAY[1..n] OF Muster;
```

nicht typkompatibel
trotz interner identischer Struktur

Zuweisungen an Feldelemente

```
name[1]        := "w";
klBuchst["f"]  := 17;
muster[blau]   := TRUE;
feld[1,3]      := 2.7;
feld[1][3]     := 2.7;
z[7, gelb]     := TRUE;
z[7][gelb]     := TRUE;
```

ARRAYs (Felder)

➢ Zuweisung, Kompatibilität

- Elementweise Zuweisung, kompatibler Typen (z.B. REAL)

```
m[i,j] := 5.0;
A[i] := B[j];
```

```
FOR i : = 1 TO n DO
  FOR j := 1 TO n DO
    IF i = j
    THEN
      identity[i,j] := 1.0
    ELSE
      identity[i,j] := 0.0
    END (* IF *)
  END (* FOR j *)
END; (* FOR i *)
```

- Sind die Elemente vom selben Typ und sind die Längen der beiden (Teil-)Felder identisch dann ist erlaubt:

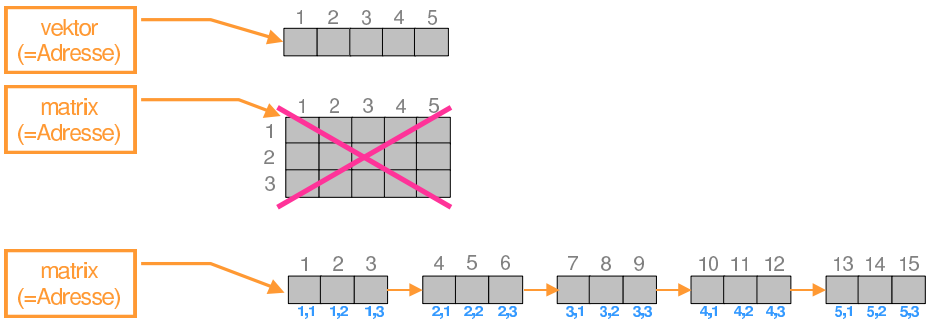
```
A := B;
```

2 Felder können nur einander zugewiesen werden, wenn sie denselben oder einen durch Typumbenennung (Alias) entstan-denen kompatiblen Typ haben!

ARRAYs (Felder)

➢ Repräsentation

```
CONST
  n = 5;
  m = 3;
VAR
  vektor : ARRAY[1..n] OF CARDINAL;
  matrix : ARRAY[1..n],[1..m] OF CARDINAL;
```

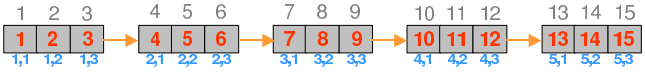


ARRAYs (Felder)

➢ Repräsentation (cont'd)

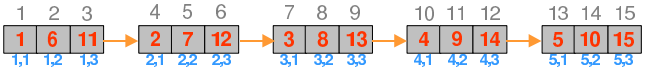
- effiziente Realisierung von Zählschleifen (aufgrund Implementation)
  - gut: innere Schleife zu 2. Index („innerer ARRAY")

```
FOR i:=1 TO n DO
 FOR j:=1 TO m DO
      matrix[i,j]:= j + (i-1)*m
 END
END
```



  - schlecht: innere Schleife zu 1. Index („äußerer ARRAY")

```
FOR j:=1 TO m DO
 FOR i:=1 TO n DO
      matrix[i,j]:= i + (j-1)*n
 END
END
```

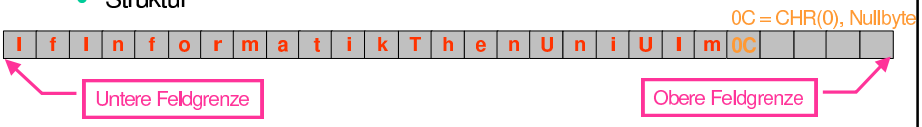


ARRAYs (Felder)

➢ Strings (Zeichenketten)

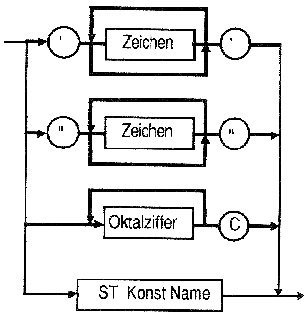
besonderer ARRAY-Typ für Komponenten vom Typ CHAR

- Struktur



- Deklaration

```
TYPE
  STRING = ARRAY[0..79] OF CHAR;

CONST
  s = "string";
  leer = ""; (* Leer-String *)
```



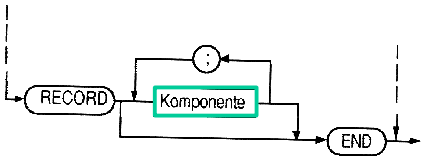
- Operationen
  Ein-/Ausgabe
  - Lesen: `FROM InOut IMPORT ReadString;`
  - Schreiben: `FROM InOut IMPORT WriteString;`

RECORDs ((Daten-)Verbunde)

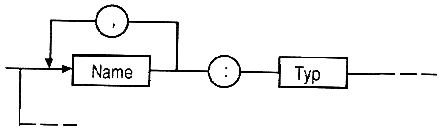
in anderen Sprachen: „structure“

- Feste Verbunde
  - Verwendung
    Zusammenfassung von Objekten verschiedener Datentypen unter einem Namen (ein Komplex)
  - Syntax

Ausschnitt aus: 15 Typ



Ausschnitt aus: 17 Komponente (KP)



RECORDs ((Daten-)Verbunde)

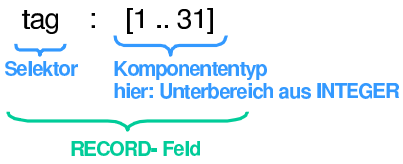
- Feste Verbunde (cont'd)
  - Deklaration (in TYPE oder VAR)
    gegeben sei ein strukturiertes Objekt (artikel) mit seinen Attributen/Komponenten (nummer, bestand, preis, lager), wobei eine Komponente wiederum strukturiert sein kann (lager → gang, regal, ebene)

```
VAR
  artikel: RECORD
             nummer  : CARDINAL;
             bestand : CARDINAL;
             preis   : REAL;
             lager   : RECORD
                         gang  : ['A'..'H'];
                         regal : [1..7];
                         ebene : [1..4]
                       END
           END
```

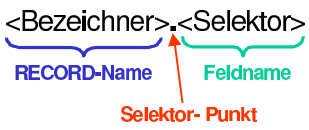
RECORDs ((Daten-)Verbunde)

➢ Feste Verbunde (cont'd)

- Komponenten
  RECORD-Feld besteht aus Selektor und Komponenten-Typ



- Zugriff auf individuelle Felder



RECORDs ((Daten-)Verbunde)

➢ Feste Verbunde (cont'd)

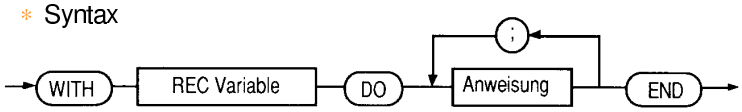
- Beispiel (1) für Zugriff und Zuweisung

```
VAR
  datum   : RECORD
              tag   : [1..31];
              monat : (Jan, Feb, Mar, Apr, Mai, Jun, Jul, Aug,
                       Sep, Okt, Nov, Dez);
              jahr  : [1900..2100]
            END

BEGIN
  ...
  datum.tag   := 10;
  datum.monat := Apr;
  datum.jahr  := 2000;

  ...
END
```

```
WITH datum DO
  tag   := 10;
  monat := Apr;
  jahr  := 2000
END;
```

RECORDs

- Zuweisung und WITH-Anweisung
  - Standard
    - Zuweisung an einzelne Elemente eines RECORD
    - Zuweisung ganzer RECORDs
  - Wiederholungs-Konstrukt (→ WITH-Klausel)
    - bisher (bei ARRAYs):
      Verarbeitung von ARRAY-Elementen mittels FOR-Anweisung
    - jetzt (für RECORDs):
      einzelne Elemente i.A. unterschiedlichen Typs (→ unterschiedliche Operationen)
      - → Zusammenfassung einer Folge individueller Anweisungen, die sich auf Felder desselben Verbundes beziehen
      - → bessere Übersichtlichkeit
        Effizienzsteigerung falls (intern!) aufwendige Index-Berechnungen notwendig
    - Syntax



RECORDs

- Zuweisung (cont'd)
  - Beispiel (2) Zuweisung

```
VAR
  heute,
  morgen,
  datum    : RECORD
               tag    : [1..31];
               monat  : (Jan, Feb, Mar, Apr, Mai, Jun, Jul, Aug,
                         Sep, Okt, Nov, Dez);
               jahr   : [1900..2100]
             END

BEGIN
  ...

  heute.jahr  := 1998;
  morgen.tag  := heute.tag + 1;
  eintrag     := heute;

  ...
END
```

Zuweisung eines ganzen RECORDs

RECORDs

- Zuweisung (cont'd)
  - Beispiel (3) Zuweisung

```
VAR
  studenten : ARRAY[0..199] OF PERSON;
  k         : CARDINAL;
  ...

BEGIN
  ...

  studenten[50].name  := "Klaus Murmann";
  studenten[50].gebdatum.monat := Jan;

  ...

  IF studenten[33].name[0] = "A"
    THEN tue_dies
    ELSE Tue_das
  ...

END
```

RECORD im RECORD