INHALTSVERZEICHNIS

1 Strukturierte Datentypen

Datentypen – Systematisierung (Zusammensetzung)
- Ein Datenyp
  - legt eine Menge von Werten (Wertebereich) sowie
  - eine Sammlung von darauf zugeschnittenen Operationen fest.

Neben einfachen Datentypen existieren in Turbo Pascal **strukturierte** Datentypen. diese setzen sich aus **einfachen** Datentypen zusammen:

- string
- array
- record
- set.

1.1 String (Zeichenkette)

- string deklariert eine Zeichenkette, d. h. eine Folge von Zeichen
- eine Zeichenkette setzt sich aus den Elementen des einfachen Datentyps charakter zusammen
- die max. mögliche Länge einer Zeichenkette beträgt 255 Zeichen.
- ist die Zeichenkette kürzer, so kann die tatsächliche Länge der Zeichenkette nach der Datentypangabe in eckigen Klammern angefügt werden.
- Bsp: VAR text: STRING [30];
- Variable text besteht aus max. 30 Zeichen, werden 40 Zeichen eingegeben, so werden nur die ersten 30 eingelesen.

(String – Zuweisungen, String-Vergleichsoperationen siehe Seite 77 Vorlesungsbeilage)

1.2 Array (Feld)

- Ein Feld (Array) besteht aus einer festgesetzten Anzahl geordneter Komponenten.
- diese Komponenten müssen alle vom gleichen Datentyp sein
- Auf die einzelenen Komponenten kann mit einem Index direkt zugegriffen werden.
- es wird unterschieden zwischen eindimensionale und mehrdimensionale Arrays

Eindimensionale Arrays

VAR scheck: ARRAY [1..25] OF real;
der Bezeichner Scheck bezieht sich auf eine Liste von 25 Variablen des Typs real;
jeder Varibalen ist eine Index-Zahl zwischen 1 und 25 zugeordnet;
Werte wurden bisher nicht vergeben, nur die Liste definiert

- VAR feld: ARRAY [1..5] OF Integer;
- VAR feld: ARRAY [´a´..´e´] OF Real;

- TYPE Werktag = (Mo, Di, Mi, Do, Fr);
  VAR Ueb_std: ARRAY [Werktag] OF Integer;
  Begin
  ....

```
Ueb_std [Mo] :=2;
...
End.
```

- Die Komponenten eines eindimensionalen Arrays werden über einen Index angesprochen
- dieser Index muß ordinalen Datentyp besitzen (z. B. integer, boolean)
- ein eindimensionales Array besteht aus einer linearen Folge von Komponenten
- Bsp.: eindimens. Array mit 10 Komponenten, das integer-Werte aufnehmen kann, wird folgendermaßen definiert.

  VAR feld: Array[1..10] of integer;

- der Zugriff auf die 3. und 7. Komponente erfolgt durch den Aufruf feld[3] bzw. feld[7]
- die Zahl in der eckigen Klammer ist der jeweilige Index der Komponente.

Zweidimensionale Arrays

- Datentyp ist nicht auf eindimensionale Arrays beschränkt
- es können auch mehrdimensionale Arrays definiert werden
- ein zweidimensionales Array kann mit einer Tabelle verglichen werden, die aus Zeichen und Spalten besteht
- Feldelemente werden über zwei Indizes direkt angesprochen (Zeilenindex und Spaltenindex)
- Beispiele: Koordinatensysteme, Matrizen und Tabellen.

- VAR matrix: ARRAY [1..3, 1..3] OF Integer;

- TYPE matrixtyp = ARRAY [1..3, 1..3] OF Char;
- VAR matrix: matrixtyp;

- Var matrix: Array [á´..´c´, á´..´c´] OF Real;

Syntax:

```
tabelle:ARRAY[1..m] OF ARRAY [1..n] OF real;
```

oder in kürzer Form:

```
tabelle:ARRAY[1..m,1..n] OF real;
```

In diesem zweidimensionalen Array besteht die 1. Zeile aus einem eindimensionalen Array mit n Komponenten, die 2. Zeile ebenso aus einem eindimensionalen Array mit n Komponenten, usw..
Das Array besteht also aus m Komponenten, wobei einzelne Komponente selber ein Array mit n Komponenten ist, d. h. es handelt sich um ein Array, dessen Komponenten selber den Datentyp array besitzen.

- wenn man auf ein Feldelement zugreifen möchte, dann muß seine Position durch beide Größen, d. h. durch Zeilen- und Spaltenangabe festgelegt werden:

  matrix [zeile, spalte]
  matrix[1,3]
  Z. B. kann mit matrix [1,3] auf den Inhald des Feldes in der 3. Zeile und 4. Spalte zugegriffen werden.

(Beispielprogramm: Matrix – siehe Seite 81 Vorlesungsbeilage)

1.3 Record (Satz)

- Ein Record enthält verschiedene Komponenten oder Felder, die aus verschiedenen Typen bestehen können.
- VAR datensatz = RECORD
  name, vorname: string [30];
  END;
- oft trifft man auf Informationen (Daten), die zusammengehören und einen Verbund bilden.
  Diese Daten sollten unter Oberbegriff zusammengefaßt werden.
  Bsp.: Name, Vorname, Straße, Hausnummer, PLZ und Ort (bilden einen Verbund)
- einzelne Daten dieses Verbunds können unterschiedlichen Datentyps sein.
  Verbund Adresse besteht aus 6 Komponenten unterschiedlichen Datentyps.
- Umsetzung so einer Zusammensetzung von Daten ist durch den strukturierten Datentyp RECORD möglich.
- RECORD besteht wie ein Feld (array) aus geordneten Komponenten, auf die direkt zugegriffen werden kann.
- im Gegensatz zu Feldern können Komponenten eines Record unterschiedliche Datentypen haben.
- RECORD enthält eine endliche Folge von Komponenten
- RECORD-Struktur beginnt mit dem Wort RECORD und endet mit dem Wort END.

- Beispiel:
  Die Record-Struktur der Variablen Pkw und lkw besteht aus
  Komponenten farbe, hubraum, tueren, hersteller, typ, mietpreis und stueck.

  Zur Übersichtlichen Gestaltung sollte eine komplexe Record-Struktur als eigenständiger Datentyp unter der TYPE-Deklaration definiert werden:

```
TYPE auto = Record
              farbe      . (rot, gruen, blau, gelb);
              hubraum    . integer;
              tueren     . 1..5;
              hersteller . (ford, vw, opel, fiat, nissan, toyota);
              typ        : string[10];
              mietpreis  . real;
              stueck     : integer;
              END;
VAR
   pkw, lkw : auto;
```

  In diesem Beispiel wird Datentyp auto definiert.
  Variablen pkw und lkw sind vom Datentyp auto.

- Der Zugriff auf eine Komponente durch Angabe des Recordnamens und Bezeichner der Komponenten.
  i.d. R.: Record-Variable, Punkt, Bezeichner im Record.

Ein weiteres Beispiel zur Erfassung der Daten einer Person:

```
TYPE adresse        =    RECORD
                         strasse          : string[30];
                         hausnr           . integer;
                         plz              . 0..99999;
                         ort              : string[20];
                         END;
      person        =    RECORD
                         vorname, name         : string[30];
                         anschrift             . adresse;
                         END;
VAR
      personalkarte  :    person;
```

Record *adresse* setzt sich aus den Komponenten: strasse, hausnr, plz und ort.
Record *person* besteht aus den Komponenten: vorname, name, die vom Typ string sind und Komponenten anschrift vom Record-Datentyp adresse.

1.4 Set (Menge)

- durch Datentyp Set werden Mengen definiert
- durch Datentyp Set werden Mengen und Operationen auf Mengen eingeführt.
- Wertebereich eines Mengentyps enthält alle denkbaren Teilmengen.
- bei n Elementen gibt es 2 hoch n Teilmengen.
- Grundmenge wird in der Deklaration explizit angegeben.
- die Grundmenge muß eine Teilmenge eines ordinalen Datentyps sein.
- Grundmenge darf höchstens 256 Elemente enthalten.

Beispiel:

```
TYPE ziffer = SET OF 0..9;
```

Grundmenge des Datentyps ziffer enthält die Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
Datentyp Ziffer beschreibt eine Menge (set), die als Elemente alle Teilmengen enthält, die sich aus den Zifern 0 bis 9 bilden lassen.
Es gibt 2 hoch 10 Teilmengen. (n=10).

Beispiel:

```
TYPE farbe    = (rot, gelb, blau);
bunt          = SET OF farbe;
```

Die Grundmenge des Datentyps bunt setzt sich aus den Elementen rot, gelb und blau zusammen.
{[rot], [gelb], [blau], [rot,gelb], [rot,blau], [gelb,blau], [rot,gelb,blau],[]}.
Oben sind 2 hoch n = 2 hoch 3 = 8 Teilmengen angegeben, wobei die leere Menge [] auch als Teilmenge definiert ist.

```
VAR
x,a,y        : ziffer;
b            : bunt;
```

Die Variablen x, a, y, und b können als Werte Elemente aus Menge ziffer bzw. bunt annehmen.
Diese sind wiederum Teilmengen, die aus den Grundmengen gebildet werden.
Folgende Wertzuweisungen sind beispielsweise möglich:
x:= [3, 5, 7..9]; Die Variable x besitzt theoretisch als Wert die Teilmenge {3,5,7,8,9}
b:= [rot, blau];

Bei Mengen aus kompatiblen Grundmengentypen können die in der Vorlesungsbeilge Seite 82 dargestellten Operationen durchgeführt werden.

Beispiel für Mengenoperationen:

[1,2,3,4] + [3,4,8,]	Ergebnis:[1,2,3,4,8]
[1,2,3,4] * [3,4,8,]	Ergebnis:[3,4]
5 IN [1,4,5,8]	Ergebnis: wahr
7 IN [1,4,5,8]	Ergebnis: falsch
[´a´..´z´] >= [´c´..´f´]	Ergebnis: wahr

IF satz [j] in vokale:
Mit IN kann überprüft werden, ob das j-te Element von satz in der Menge vokale enthalten ist.

2. Erweiterungen in Pascal

In Pascal ist es möglich, zwischen ***statischen*** und ***dynamischen Datenstrukturen*** zu unterscheiden.
Einfache bzw. strukturierte Datentypen sind ***statisch***, da sie sich in Form und Grö0e nicht ändern.
Bei der Compilierung wird für diese Datentypen der Speicherbedarf unveränderlich festgelegt.
Beispiel hierfür ist das ***Array***. Die Anzahl seiner Komponenten sind von vornherein bestimmt, um notwendigen Speicherplatz vorab reservieren zu können.

Bei vielen Anwendungen läßt sich die Anzahl der Komponenten nicht im voraus festlegen. Wird ein Array nicht vollständig belegt, so bedeutet dieses ***ineffiziente Speichernutzung***.
Andererseits kann ein Programm mehr Daten erzeugen, als vorab deklarierte Array aufnehmen kann, d. h. reservierte Speicherplatz reicht nicht aus.

Abhilfe hierfür bieten die ***dynamischen Datenstrukturen***.
Hier wird der Speicherplatz erst während des Programmablaufs zugewiesen, da diese Datantypen sich in iherer Struktur verändern können.
Zu diesen Dynamischen Datenstrukturen zählen ***Datentyp File (Datei)*** sowie der Datentyp ***Pointer (Zeiger)***.
Pointer wird z. B. zum Aufbau linearer und nihtlinearer Listen genutzt.

2.1 Prozeduren und Funktionen

- sie bieten die Möglichkeit, zusammengehörige Anweisungsfolgen unter einem Namen zusammenzufassen;
- durch sie können Programme in Teile zerlegt und übersichtlicher gestaltet werden;
- Unterprogrammtechnik;
- Einzelne abgrenzbare Aufgaben werden aus dem Hauptprogramm ausgegliedert
- Wiederverwendung von Programmmodulen;
- Datenaustausch zwischen Haupt- und Unterprogrammen über Parameter.

2.1.1 Prozeduren

- Ein Programm läßt sich gliedern in Unterprogramm(e) und Hauptprogramm.
- Eine Porzedur ist ein Unterprogramm und fasst mehrere Anweisungen unter einem Namen zusammen. Sie werden im Hauptprogramm über diesen Namen aufgerufen
- Der Aufruf einer Prozedur im Hauptprogramm bzw. in einem weiteren Unterprogramm . erfolgt als einzelne Anweisung und darf nicht innerhalb eines Ausdrucks erfolgen.
- Im Deklarationsteil muß eine solche Prozedur (und auch Funktion) vereinbart werden.
- Die Reihenfolge sollte folgendermaßen eingehalten werden:

1. Konstantendeklaration,
2. Datentypdeklaration,
3. Variablendeklaration,
4. Prozedur- und Funktionsdeklaration.

Prozedurdeklaration besteht aus:

Prozedurkopf (Wortsymbol procedure, Prozedurnamen und Deklaration von formalen Parametern)
Prozedurblock (Deklarations- und einen Anweisungsteil)

Damit besitzt eine Porzedure denselben Aufbau wie ein Programm.
Abweichend ist das einleitende Wort „procedure“ anstatt „program“ und der Abschluß mit ´;´anstelle von ´.´ zu beachten.

- alle im Deklarationsteil eines Prozedurblocks vereinbarten Variablen haben Gültigkeit für diesen und für alle untergeordneten Blöcke.
- außerhalb des Prozedurblocks sind sie nicht bekannt, weshalb sie auch lokale Variablen genannt werden.
- Die in einem übergeordneten Prozedurblock deklarierten Variablen sind für die untergeordneten Blöcke global, d. h. sowohl innerhalb der Blöcke als auch in der übergeordneten Prozedur bekannt.
- Nach der Prozedurdeklaration folgt das Hauptprogramm, in welchem die Prozedur über ihren Namen aufgerufen wird.
- Prozeduren können wiederum andere Prozeduren aufrufen, wobei eine Prozedur, die von einer anderen aufgerufen wird, vor dieser deklariert sein muß.
- Mehrmalige Aufruf einer Prozedur in einem Hauptprogramm ist erlaubt.
- Prozeduren sind häufig umfangreiche Teilprogramme.

2.1.2 Funktionen (S.89,90 VB)

- Sie sind neben Porzeduren eine weitere Form von Unterprogrammen.
- Der Unterschied liegt darin, daß Funktionen immer einen konkreten Funktionswert an das aufrufende Hauptprogramm zurückrückliefern.
- Das über Prozeduren gesagte gilt entsprechend für Funktionen.
- Aufruf einer Funktion kann im Hauptprogramm bzw. in einem weiteren Unterprogramm innerhalb eines Ausdrucks erfolgen.
- In einem Programm muß die Prozedurdeklaration vor der Funktionsdeklaration erfolgen.
- Funktionsdeklaration besteht aus:
  Funktionskopf (Wortsymbol function, Funktionsname, Parameterdeklaration) und einem Funktionsblock (Deklarations- und Anweisungsteil)

Funktionen und Prozeduren – VORTEILE

1. Durch die Verwendung von Prozeduren und Funktionen werden Programme ***übersichtlicher***, da Programmteile unter einem eigenen Namen zusammengefaßt werden können.
2. Mittels Prozedur wird es möglich, eine Folge von Anweisungen ***mehrmals*** an verschiedenen Stellen eines Programms ***ausführen*** zu lassen. ***Programmieraufwand*** sinkt.
3. ***Wartungsaufwand*** reduziert sich, da Änderungen nur an einer Stelle durchgeführt werden müssen.

Globale und lokale Variablen

- Der Gültigkeitsbereich der Bezeichner ist abhängig von der jeweiligen Deklaration der Bezeichner in den verschiedenen Programmblöcken (Hauptprogramm, Unterprogramme)
- Alle in ***einem Block (Prozedur oder Funktion)*** vereinbarten Variablennamen gelten nur in diesem und in den untergeordneten Blöcken. Sie sind für diesen Block ***lokal***.
  Lokale Variable außerhalb des Blocks nicht vorhanden.
- In einem ***übergeordneten Block*** vereinbarten Variablen sind für die untergeordneten Blöcke ***global***.
- Wenn lokale und globale Größe denselben Namen haben, so ist immer die lokale Größe gemeint.
  Der Wert der globalen Größe dieses Namens wird für die Dauer des Blocks ***„eingefroren“***.

Wertparameter (S.92 VB)

- Wertparameter sind Parameter, die in der Parameterliste einer Prozedur oder Funktion nur mit Ihrem Namen aufgeführt sind.
  Bei Wertparametern wird der Prozedur oder Funktion nur eine Kopie des Wertes übergeben.
- Veränderungen der Parameter betreffen nur diese Kopie. Der ursprüngliche Wert innerhalb des übergeordneten Blocks bleibt erhalten.
- Beispiel siehe S. 92 Vorlesungsbeilage.

Referenzparameter (S. 92 VB)

- Referenzparameter sind Prameter, die in der Parameterliste mit dem Zusatz VAR aufgeführ werden.
- Bei Referenzparametern wird der Prozedur oder Funktion die Speicheradresse der Variable übergeben.
- Veränderungen an Parametern innerhalb der Prozedur oder Funktion verändern auch den Originalwert des übergeordneten Blocks.
- Beispiel siehe S. 92 ,93Vorlesungsbeilage.