Agenda für heute, 18. März, 2010

- **Strukturierte Variablen**
- Arrays
- Typendefinitionen
- Typische Operationen mit Arrays
- Strings

Lesen Sie den Begleittext Seite 40 - 43

Was sind strukturierte Variablen?

- Eine Variable eines strukturierten Datentyps kann mehr als eine Komponente enthalten.

```
var  S: array[1..5]of char
```

S

S[1]
S[2]
S[3]
S[4]
S[5]

- Jede Komponente eines strukturierten Datentyps ist eine Variable, die von einem einfachen (siehe S) oder strukturierten Datentyp sein kann (siehe T).

```
var  S: array[1..5]of string
```

T

	T[1,2]		
T[4,1]			

T[1,2]: char
T[4,1]: string [1]
T[4]: string

Eigenschaften strukturierter Variablen

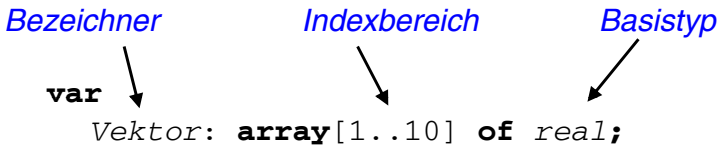
- Ein einziger Bezeicher erlaubt Bezugnahme auf mehr als einen Datenwert.
- Indizierte Variablen können mit Schleifen verarbeitet werden.
- Die Komponenten auf der untersten Stufe einer strukturierten Variablen sind von einem einfachen Datentyp und können wie einfache Variablen verwendet werden.
- Strukturierte Datentypen in Pascal sind:
  - Array
  - String
  - Record
  - File

- Strukturierte Variablen
- **Arrays**
- Typendefinitionen
- Typische Operationen mit Arrays
- Strings

Strukturierte Datentypen: Der Array

Eine Variable vom Typ Array erlaubt Daten eines einheitlichen Datentyps so zu speichern, dass auf sie über einen Index zugegriffen werden kann.

Er ist definiert durch einen Indexbereich und den Datentyp der Elemente.



Eigenschaften von Arrays

Indexbereich

- Ordinaltyp
  (geordnete Menge)
- Literale
- Konstanten

Basistyp

- Standardtyp
  (integer, real, char etc.)
- einfacher Typ
- strukturierter Typ

Zuweisungsanweisungen mit Arrays

```
u, v: array['a'..'z'] of integer;
```

Die Anweisung `u:= v`

ist gleichbedeutend mit der Anweisungsfolge

```
u['a']:= v['a'];
u['b']:= v['b'];
u['c']:= v['c'];
etc.
```

Ebenfalls möglich: `u[chr(103)]:= v['g'];`

Arraydeklarationen

Wenn in Pascal Arrays wie folgt deklariert werden:

```
var   u: array[1..3] of real;
      v: array[1..3] of real;
```

dann ist die Anweisung

```
u:= v
```

nicht möglich, wohl aber

```
u[1]:= v[1];
```


- Strukturierte Variablen
- Arrays
- **Typendefinitionen**
- Typische Operationen mit Arrays
- Strings

Typendefinitionen

In Pascal können neben den Standardtypen `Integer`, `Real`, `Char` und `Boolean` eigene Datentypen definiert werden.

Dazu wird ein Typendefinitions-Abschnitt eingeführt:

```
const  N = 10;

type   Buchstabe = 'a'..'z';
       Teenager = 1..19;
       t = array[1..N] of real;

var    Junge: Teenager;
       Alte: 70..110;
```


Arraydeklarationen kombiniert mit Typendefinitionen

In Pascal können Arrays aber auch wie folgt deklariert werden:

```
type  t = array[1..3] of real;

var   u: t;
      v: t;
```

dann sind beide Anweisungen

```
u:= v;
```

und

```
u[1]:= v[1];
```

möglich

- Strukturierte Variablen
- Arrays
- Typendefinitionen
- **Typische Operationen mit Arrays**
- Strings

Typische Operationen mit Arrays

```
const
  N = 200;
var
  MessWerte: array [1..N] of real;
  i: integer;  sum: real;
```

Initialisieren:

```
for i:= 1 to N do
  MessWerte[i]:= 0
```


Typische Operationen mit Arrays

Summieren:

```
sum:= 0;
for i:= 1 to N do
sum:= MessWerte[i] + sum
```


Einfache Algorithmen mit Arrays

```
var
  MessWerte: array [1..N] of real;
  i, k: integer;  min: real;
```

Minimum suchen:

```
for i:= 1 to N do read(MessWerte[i]);
k:= 1;
min:= MessWerte[k];
for i:= 2 to N do
  if MessWerte[i] < min
    then begin
           k:= i;
           min:= MessWerte[k]
         end
```

min enthält den kleinsten Wert, ***k*** dessen Position im Array

Minimum suchen, Beispiel

Wert	8	12	5	17	2	9	26	6	15	11
Index *i*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

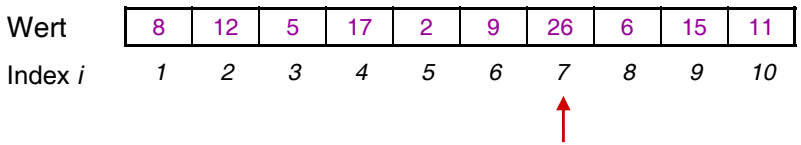
Variablenwerte

Durchlauf	*i*	*k*	*min*	*MessWerte*[*i*]
1	2	1	8	12
2	3	3	5	5
3	4	3	5	17
4	5	5	2	2
5	6	5	2	9
6	7	5	2	26
7	8	5	2	6
8	9	5	2	15
9	10	5	2	11

Einfache Algorithmen mit Arrays

Lineares Suchen eines Wertes in einem Array

Im folgenden Array *MessWerte* soll der Wert 26 gesucht werden:




```
i:= 1;
x:= 26;
while MessWerte[i] <> x do
   i:= i + 1
```

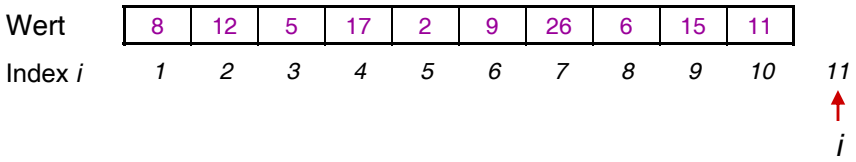
Was passiert, wenn der Wert 4 gesucht wird?

Der Algorithmus "schiesst" über die Arraygrenze hinaus! = Laufzeitfehler

Einfache Algorithmen mit Arrays

Lineares Suchen, korrekte Version:

```
i:= 1;  x:= 4;  N:= 10;
  while (i <= N) and (MessWerte[i] <> x) do
    i:= i + 1
```



Funktioniert allerdings nur mit dem *Kurzschlussverfahren* bei der Auswertung Boolescher Ausdrücke:

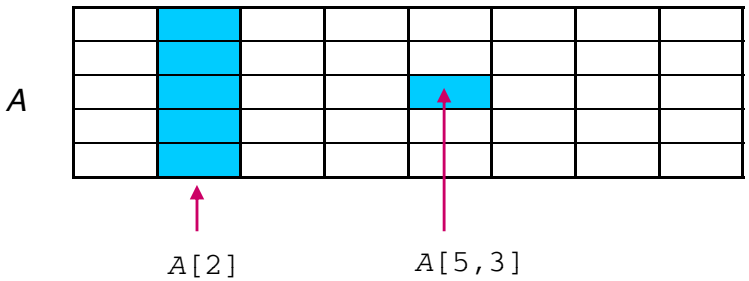
Wenn der Vergleich `(i <= N)` false ergibt
dann wird `MessWerte[i] <> x` nicht ausgewertet.

Zweidimensionale Arrays

Arrayelemente können auch von einem strukturierten Typ sein.

Damit lassen sich mehrdimensionale Strukturen verwalten (z.B. Matrizen)

```
Var
  A: array[1..8] of array [1..5] of integer;
```



Verschiedene Deklarationen für 2-d Arrays

a)
```
var
  B: array [1..N] of array ['a'..'z'] of integer;
```

b)
```
var
  B: array [1..N,'a'..'z'] of integer;
```

c)
```
type
  C = array ['a'..'z'] of integer;
var
  B: array [1..N] of C;
```

2-d Arrays referenzieren

```
const N = 10;

var   i: integer;
      ch: char;
      B: array [1..N,'a'..'z'] of integer

for i:= 1 to N do
  for ch:= 'a' to 'z' do read(B[i,ch]);
```


- Strukturierte Variablen
- Arrays
- Typendefinitionen
- Typische Operationen mit Arrays
- **Strings**

Eigenschaften von Strings

```
var S: string[14];
S:= 'ETH Zuerich';
```