

ARRAYS (Felder)

Sonderübung

Luise Zieger & Lea Happel

PROG Sonderübung

December 18, 2018

1. Wiederholung
2. ARRAYS (Felder)
 - Statische ARRAYS
 - Standardfunktionen
 - Dynamische ARRAYS
 - Vordefinierte Funktionen
3. Übungsaufgaben
4. Felder in Unterprogrammen
 - Felder als UP-Parameter
 - Felder als Funktionsergebnis
 - Automatische Felder

Wiederholung: Ganz grundlegende Sachen über Arrays

Allgemeine Syntax eines Arrays mit fester Größe

`<Datentyp>, DIMENSION($a_1 : e_1, a_2 : e_2, \dots, a_N : e_N$) :: Array1`

! Zugriff auf einzelne Felder

`Feldeintrag=Array1(i_1, i_2, \dots, i_n)`

! Zugriff auf Teilfelder

`Feldeintrag=Array1($i_1 : j_1, i_2 : j_2, \dots, i_n : j_n$)`

- In Fortran **beginnen Arrays (per default) immer mit 1!**
- Ein 1-dimensionales Array (entspricht Vektor) kann man mit dem Feldkonstruktor vorbelegen
- Um ein mehrdimensionales Array vorzubelegen, erzeugt man zuerst ein eindimensionales Array mit dem Feldkonstruktor und wandelt dieses mit `reshape` um

```
INTEGER, DIMENSION(1:3):: A
```

```
A=(/1,2,3/)
```

ARRAYS (Felder)

- homogene Datenstruktur (alle Elemente haben denselben Typ)
- ein- oder mehrdimensional (Vektor, Matrix, Tensor, ...)
- Feldtyp charakterisiert durch **Elementtyp** (bel. skalarer Typ) und **Rang** (bzw. rank) = Anzahl der Dimensionen (≤ 15)
- entweder **statisch** oder **dynamisch**

Deklaration statischer ARRAY-Variablen

```
< Elementtyp >, DIMENSION ([ $l_1$  :]  $u_1$ , ... , [ $l_r$  :]  $u_r$ ) :: my_array
```

```
r := rank
```

Beispiel

```
INTEGER, DIMENSION (5) :: vector
```

! Vektor der Länge 5

```
INTEGER, DIMENSION (1:4, 1:4) :: matrix
```

! 4 × 4 - Matrix

Standardfunktionen

INTEGER, DIMENSION (5) :: vector ! Vektor der Länge 5

INTEGER, DIMENSION (1 : 4, 1 : 4) :: M ! 4 × 4 - Matrix

Ausdehnung:

- Untergrenze (Default-Untergrenze: 1)
- Obergrenze

Standardfunktionen:

LBOUND (M, i)	$=: l_i$	Untergrenze in der i-ten Dimension
UBOUND (M, i)	$=: u_i$	Obergrenze in der i-ten Dimension
LBOUND (M)	$= (/l_1, l_2, \dots, l_r/)$	liefert Vektor
UBOUND (M)	$= (/u_1, u_2, \dots, u_r/)$	
SIZE (M, i)	$= u_i - l_i + 1 =: d_i$	Anz. Indexwerte in der i-ten Dimension
SIZE (M)	$= \prod_{i=1}^r \text{SIZE (M, i)}$	
SHAPE (M)	$= (/d_1, d_2, \dots, d_r/)$	

Deklaration dynamischer ARRAY-Variablen

```
< Elementtyp >, DIMENSION ( : , : , ... , : ), ALLOCATABLE :: my_array
```

Beispiel

```
INTEGER, DIMENSION (:), ALLOCATABLE :: dyn_vec  
INTEGER, DIMENSION ( : , : ) :: dyn_mat, A, B
```

Dynamische Felder müssen **allokiert** werden. D. h. die Gestalt des Feldes wird festgelegt und entsprechend Speicherplatz beschafft.

```
ALLOCATE (dyn_mat(4, 4), dyn_vec(0 : 6))
```

Das Feld wird **deallokiert**, d. h. der Speicherplatz wird wieder freigegeben.

```
DEALLOCATE (dyn_mat, dyn_vec)
```

Vordefinierte Funktionen

INTEGER, DIMENSION (3 , 4) :: M

! 2 - dim. Feld

SUM (M, i), i = 1, 2

Vektor der Spalten- bzw. Zeilensummen

$$\text{SUM (M)} = \sum_i \sum_j m_{ij}$$

Summe aller Matrixeinträge

PRODUCT (M, i), i = 1, 2

Vektor der Spalten- bzw. Zeilenprodukte

$$\text{PRODUCT (M)} = \prod_i \prod_j m_{ij}$$

Produkt aller Matrixeinträge

MINVAL (M, i)

kleinster Eintrag

MAXVAL (M, i)

größter Eintrag

MINLOC (M, i)

Position des kleinsten Eintrags

MAXLOC (M, i)

Position des größten Eintrags

Vordefinierte Funktionen

INTEGER, DIMENSION (4 , 4) :: M, A, B

LOGICAL, DIMENSION (2 , 3) :: L

LOGICAL, DIMENSION (4) :: v, w

TRANPOSE (M) Liefert transponierte Matrix (für 2-dim. Felder!)

MATMUL (A, B) = $A \cdot B$

MATMUL (A, w) = $A \cdot w$

MATMUL (v, B) = $v^T \cdot B$

DOT_PRODUCT (v, w) Skalarprodukt ($\sum_i v_i \cdot w_i$)

Für LOGICAL-Felder:

ANY (L [, i]) Logisches ODER (.TRUE., falls **ein** Eintrag .TRUE.)

ALL (L [, i]) Logisches UND (.TRUE., falls **alle** Einträge .TRUE.)

Vordefinierte Funktionen

- alle intrinsischen Operatoren ($+$, $-$, $*$, $/$, ...) und die meisten intrinsischen Funktionen (z. B. ABS) auf Felder anwendbar (sofern der Elementtyp stimmt)
- dabei wird eintragsweise gerechnet
- ACHTUNG: Operanden (bei binären Operatoren) müssen gestaltkonform sein
- Skalare sind mit bel. Feldern gestaltkonform

z. B. $3 + v$, $v + w$, $v * w$, $v == w$, ANY ($v == w$), ...

Übungsaufgabe

Wetterstation

Eine Wetterstation hat für 14 Tage folgende Temperaturwerte aufgenommen.

Tag	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Temp.	12	14	9	12	15	16	15	15	11	8	14	13	15	12

Speichere die Tabelle mit einem geeigneten Datentyp. Schreibe ein Programm, welches die Durchschnittstemperatur sowie die maximale und minimale Temperatur für die zwei Wochen bestimmt. Zudem soll berechnet werden, welche die beiden aufeinanderfolgenden Tage sind, an denen es den größten Temperaturumschwung gab. Gib anschließend die Tabelle sowie die ermittelten Werte aus.

Übungsaufgabe

SUDOKUS

Wir möchten testen, ob 2 × 2 SUDOKUS richtig gelöst wurden.

Für ein allgemeines SUDOKU der Größe $n^2 \times n^2$ gelten die folgenden Regeln:

- in jeder Zeile taucht jede der Zahlen $1, \dots, n^2$ genau einmal auf
- in jeder Spalte taucht jede der Zahlen $1, \dots, n^2$ genau einmal auf
- in jedem Block (der Größe $n \times n$) taucht jede der Zahlen $1, \dots, n^2$ genau einmal auf

Folgende SUDOKUS sollen getestet werden:

1	2	3	4	1	2	3	4	
3	4	1	2	2	1	4	3	...
2	3	4	1	1	2	3	4	
4	1	2	3	2	1	4	3	

Felder in Unterprogrammen

Felder als UP-Parameter

- statisches Feld oder
- Feld übernommener Gestalt: Gestalt zum Aufrufzeitpunkt des UPs durch a. A. festgelegt

```
FUNCTION mult (A, v)
  INTEGER, DIMENSION ( : , : ) :: A
  INTEGER; DIMENSION (0 : ) :: v
  ...
```

Felder in Unterprogrammen

Felder als Funktionsergebnis

Indexgrenzen müssen zum Aufrufzeitpunkt des UPs berechenbar sein. Möglich: Ausdrücke, die von d. UP-Parametern abhängen, z. B.

```
FUNCTION mult (A, v)
  INTEGER, DIMENSION ( : , : ) :: A
  INTEGER, DIMENSION ( 0 : ) :: v
  INTEGER, DIMENSION ( 1 : SIZE (A, 1)) :: mult
```

Felder in Unterprogrammen

Automatische Felder

- lokale Variablen des UPs
- Indexgrenzen müssen zum Aufrufzeitpunkt des UPs berechenbar sein, können von d. UP-Parametern abhängen

```
FUNCTION mult (A, v)
  INTEGER, DIMENSION ( : , : ) :: A
  INTEGER, DIMENSION ( 0 : ) :: v
  INTEGER, DIMENSION ( 1 : SIZE (A, 1)) :: mult
  INTEGER, DIMENSION ( SIZE (A, 2), SIZE (A, 1) ) :: T
  ...
  T = TRANSPOSE (A)
  ...
```