

# IO und Formatangaben

## Sonderübung

Robin Flemming & Jonas Riedel

PR10

17.01.2023

1. Wiederholung: Felder in Unterprogrammen
  - Felder als UP-Parameter
  - Felder als Funktionsergebnis
  - Automatische Felder
2. Übungsaufgabe
3. Input und Output
  - Öffnen von Dateien
  - Schließen von Dateien
  - Besonderheiten von Scratch-Dateien
  - Lesen und Schreiben in Dateien
4. Übungsaufgabe

# Felder in Unterprogrammen

## Felder als UP-Parameter

- statisches Feld oder
- Feld übernommener Gestalt: Gestalt zum Aufrufzeitpunkt des UPs durch aktuelles Argument festgelegt

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

```
END FUNCTION
```

# Felder in Unterprogrammen

## Felder als Funktionsergebnis

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

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

# Felder in Unterprogrammen

## Automatische Felder

- lokale Variablen des UPs
- Indexgrenzen müssen zum Aufrufzeitpunkt des UPs berechenbar sein, können von den 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)
  ...
END FUNCTION
```

# Übungsaufgabe

Schreibe ein Hauptprogramm, das

- die Dimension der Matrix einliest
- eine quadratische Matrix dieser Dimension allokiert

Schreibe ein Modul mit

- einer Subroutine zum zeilenweisen Einlesen einer Matrix übernommener Gestalt
- einer Funktion zur Berechnung der Spur (Summe der Diagonaleinträge) einer Matrix
- testet, ob die Matrix orthogonal ist ( $AA^T = I$ )
- den Positivteil der Matrix zurückgibt

# Aufgabe und Nutzen von IO

## Aufgaben:

- Verwaltung von Dateien
- Datentransfer zwischen Dateien und internem (Haupt-)Speicher

## Nutzen:

- Speichern von Ergebnissen über den Programmablauf hinaus
- vereinfacht das Einlesen großer Datenmengen (d.h. man muss nicht alles selbst tippen)

# Öffnen von Dateien

```
OPEN([UNIT=]u [,IOSTAT=iostat] [,ERR=err][,FILE=file]  
[,STATUS=status][,ACTION=action])
```

- wähle immer eine Unitnummer **> 30**, um keine bereits intern vergebenen Nummern zu nutzen
- Bei File sollte entweder eine Charaktervariable stehen oder Dateiname in ' '
- STATUS ( OLD/NEW/REPLACE/SCRATCH/ **UNKNOWN**) und ACTION (READ/WRITE/**READWRITE**) dienen dazu, euch vor euch selbst zu schützen  $\Rightarrow$  ungewolltes Überspeichern/ Verändern von Dateien verhindern
- wenn ihr eine IOSTAT-Variable habt, dann fragt sie auch ab!

## Beispiel

```
INTEGER:: iopen  
DO  
    OPEN (UNIT=31,IOSTAT=iopen, FILE='Beispiel.dat', &  
          & STATUS='new', ACTION='write')  
    IF (iopen /= 0) EXIT  
END DO
```



# Schließen von Dateien

```
CLOSE([UNIT=u [,IOSTAT=iostat] [,ERR=err][,FILE=file][,STATUS=status])
```

- Meistens verwenden wir bei CLOSE nur die Unitnummer und keine Attribute

## Beispiel

```
CLOSE(UNIT=31)
```

```
CLOSE(31)
```

# Besonderheiten von Scratch-Dateien

Scratch-Dateien sind (unbenannte) temporäre Dateien, die bei der OPEN-Anweisung erzeugt werden und bei der Ausführung von CLOSE/ beim Beenden des Programms gelöscht werden.

- Nützlich z. B. zum Speichern von Nebenrechnungen/Zwischenergebnissen
- Beim Öffnen von Scratch-Dateien darf kein Name (FILE-Attribut) vergeben werden
- Beim Schließen darf nicht STATUS='keep' verwendet werden

## Beispiel

```
INTEGER:: iopen
```

```
OPEN(UNIT=32,IOSTAT=iopen, STATUS='scratch')
```

# Lesen und Schreiben in Dateien

Bei READ(\*,\*) und WRITE(\*,\*) steht der Eintrag vor dem Komma für **'Woher?'** und der danach für **'Was?'**.

Ausführlich: READ(**UNIT=\*,FMT=\***) bzw. WRITE(**UNIT=\*,FMT=\***)

\* steht für Standard, d.h. beim Lesen die Tastatur, beim Schreiben der Bildschirm

Um in eine Datei zu schreiben/ aus einer Datei zu lesen, ersetzt ihr das \* bei Unit durch die entsprechende Unitnummer.

Wenn nicht anders spezifiziert, geschieht lesen/schreiben zeilenweise!

## Beispiel

```
INTEGER:: iopen
```

```
OPEN(UNIT=31,IOSTAT=iopen, FILE='Beispiel.dat', STATUS='new', ACTION='write')
```

```
IF( iopen /= 0) STOP
```

```
WRITE(31,*) 'Hallo'
```

```
WRITE(31,*) 'Welt!'
```

# Übungsaufgabe

In einer Datei steht in der ersten Zeile, wie viele Zeilen noch folgen und wie viele Zeichen (Spalten) diese jeweils enthalten. Danach ist der Text aber senkrecht statt waagrecht geschrieben, Leerzeichen sind mit \_ dargestellt.

Um dies zu lesen, öffne zuerst die Datei und lies die erste Zeile. Allokiere dann eine Matrix mit Einträgen vom Typ Character in entsprechender Größe.

Lies nun den Text zeilenweise in eine Matrix ein und schreiben ihn dann spaltenweise auf den Bildschirm und in eine extra Datei.

ZUSATZ: Schreibe eine Funktion, die eine entsprechende Textdatei erstellt. Den Text kannst du zeilenweise eingeben lassen. Beachte jedoch, dass in der Datei zwischen jedem Zeichen ein Leerzeichen stehen muss.

```
4 6
H O R N U E
A _ T ! F H
L F R _ _ T
L O A A G S
```