

# Hallo FORTRAN

## Sonderübung

Emily Gasmann & Anneli Richter

PR10

25.10.2021

1. Wiederholung
  - Grundlegende Struktur eines FORTRAN-Programms
2. Variablen
3. Datentypen
  - INTEGER
  - REAL
    - Übungsaufgabe
  - CHARACTER
    - Übungsaufgabe
  - LOGICAL
4. Attribute
5. Übungsaufgabe
6. Zusammenfassung

# Wiederholung

```
PROGRAM taschenrechner  
  IMPLICIT NONE  
  
  INTEGER :: a, b, ergebnis  
  
  READ(*,*) a, b  
  ergebnis = a + b  
  WRITE(*,*) "a + b = ", ergebnis  
END PROGRAM
```

# Grundlegende Struktur

## IMPLICIT NONE

- IMPLICIT NONE Statement sollte verwendet werden
- wird IMPLICIT NONE nicht verwendet, wird der Variable "zufällig" (nach Lage des Buchstabens im Alphabet) ein Datentyp zugewiesen (  $\Rightarrow$  Chaos)

## Maximale Zeilenlänge

- beträgt 132 Zeichen
- ACHTUNG: Ist eine Rechnung mehr als 132 Zeichen lang, jedoch bis zum 132. Zeichen syntaktisch korrekt, wird diese missverstanden, da nur die ersten 132 Zeichen übersetzt werden!

# Grundlegende Struktur

## Zeilenumbruch

Was ist wenn man mehr als eine Zeile braucht? Dann hilft das kaufmännische UND (&), welches wie eine "Klebelasche" wirkt.

```
output = input1 + input2
```

! Möglicher Zeilenumbruch

```
output = input1 &  
+ input2
```

! Eine andere Möglichkeit des Zeilenumbruchs

```
output = input1 &  
& + input2
```

Zeilenfortsetzungen dürfen sich auf maximal 40 Zeilen erstrecken.

# Grundlegende Struktur

## Groß-/Kleinschreibung

- Seit dem Fortan 90 Standard ist Fortran case **ins**ensitive
- d. h. alles (Befehle, Variablennamen, Funktionsnamen etc.) kann groß wie auch klein geschrieben werden; es ist identisch
- ACHTUNG: var und Var bezeichnen die gleiche Variable (im Gegensatz zu anderen Programmiersprachen)

```
prOgraM caSE  
  imPLlCit nONe  
  
eND ProGrAm caSE
```

Dieses Programm ist möglich, jedoch offensichtlich unleserlich.

## HINWEIS

Sucht euch für die Schlüsselwörter einen Stil aus, groß oder klein, und setzt diesen durch. Denn: Konsistenz erhöht die Leserlichkeit!

# Grundlegende Struktur

## Kommentare

- Zeilen oder Abschnitte, die bei Übersetzung des Programms ignoriert werden
- hilfreich, wenn der Source Code sehr lang ist oder verschiedene Programmierer gemeinsam an einem Projekt arbeiten oder um bei der Abgabe auch noch zu wissen, warum man was wie gemacht hat
- in Fortran95 (sowie Fortran90, Fortran03, Fortran08): mit einem ! eingeleitet
- Rest der Zeile vom Compiler beim Übersetzen nicht beachtet

```
PROGRAM commentsexample
```

```
  implicit none
```

```
  ! Diese Zeile ist ein Kommentar
```

```
  WRITE(*,*) "Das ist Text" ! Das ist ein Kommentar, Befehle und Kommentare  
  ! können in der gleichen Zeile stehen.
```

```
  !! ein 2. Ausrufezeichen macht nichts, alles ab dem 1. AZ ist Kommentar
```

```
  ! WRITE(*,*) "Dieser Text wird nicht geschrieben" ! Dieser Befehl steht  
  ! in einem Kommentar und wird deswegen ignoriert
```

```
END PROGRAM commentsexample
```

# Was ist eine Variable?

- mögliche Interpretation: Variable = Platzhalter oder Box
- Box hat Platz für **genau** eine Sache
- Box hat einen **Namen**, mit dem wir die Box ansprechen, ihren **Inhalt** lesen und ihr einen neuen Inhalt geben können
- Form und Größe der Box vorgegeben (durch einen **Datentyp** beschrieben), Inhalt der Box (bzw. Variable) muss zu diesem Datentypen passen

## Regeln für Variablennamen:

- maximal 31 Zeichen
- muss mit einem Buchstaben beginnen
- darf Buchstaben, Ziffern und Unterstriche (\_) beinhalten



# Variablendeklaration

- in F95 müssen alle Variablen am Anfang des Programs deklariert werden
- `< Datentyp > :: < NameVariable1 > , < NameVariable2 >`

```
PROGRAM deklaration  
    IMPLICIT NONE  
  
    INTEGER :: a, z  
    CHARACTER :: anfang, ende  
END PROGRAM
```

# INTEGER

## INTEGER

- repräsentiert einen Teil der ganzen Zahlen  $\mathbb{Z}$
- Wertemenge  $W = \{-2^{k-1}, -2^{k-1} + 1, \dots, 2^{k-1} - 1\}$  (k-bit Darstellung im 2er Komplement)
- können i. A. positiv oder negativ sein
- Operatoren: +, - (jeweils unär und binär), \*, /, \*\*
- **Aufgepasst:  $4/5 * x = 0$  für alle x**

# REAL

## REAL

- Wertebereich  $W = R(b, l, \underline{e}, \bar{e})$  definiert durch GK-Format
- Operatoren:  $+$ ,  $-$  (jeweils unär und binär),  $*$ ,  $/$ ,  $**$

Nützliche generische, intrinsische Funktionen für **REALs**:

- $KIND(X)$  gibt den KIND-Wert zurück
- $RADIX(X)$  gibt die Basis des Zahlenmodells derjenigen Zahlen an, die den gleichen Typ und Typparameterwert wie X haben
- $MINEXPONENT(X)$ ,  $MAXEXPONENT(X)$  geben den kleinsten bzw. größten möglichen Exponenten des Zahlentyps von X an
- $TINY(X)$ ,  $HUGE(X)$  geben die kleinste bzw. größte Zahl des Datentyps von X an
- $DIGITS(X)$  gibt die Anzahl der möglichen signifikanten Stellen von X an
- $EPSILON(X)$  gibt eine positive Modellzahl, die fast vernachlässigbar ist gegenüber der Einheit in dem Zahlenmodell von X (Wilkinson-Epsilon)
- $RANGE(X)$  gibt den dezimalen Exponentenbereich des Zahlenmodells von X an
- $PRECISION(X)$  gibt dezimale Genauigkeit reeller Zahlen mit gleichem Typ und Typparameterwert wie X an

# Aufgaben

- ❶ Schreibt ein neues Programm pi.f95, welches zu einem eingelesenen Radius den Kreisumfang berechnet. Deklariert dazu eine Variable pi vom Typ REAL und weist dieser den Wert 3.14 zu. Das Ergebnis soll kommentiert ausgegeben werden.
- ❷ **ZUSATZ:** Testet die intrinsische Funktion HUGE mit
  - a) einer INTEGER-Variablen und
  - b) einer REAL-Variablen.Welcher Wert wird jeweils zurückgegeben?

# CHARACTER

## **CHARACTER[(len = k)]**

- bezeichnen Zeichenketten
- ohne len-Angabe: len = 1 (Default-Wert)

**CHARACTER :: one\_letter** ! Die Variable one\_letter besteht nur aus einem Zeichen

**CHARACTER(len=30) :: many\_letters**

! Die Variable many\_letters besteht aus 30 Zeichen

**many\_letters = "not 30 characters yet"** ! Das sind noch keine 30 Zeichen

! aber many\_letters wird mit Leerzeichen aufgefüllt.

**WRITE(\*,\*) many\_letters** ! Gibt alle 30 Zeichen auf der Konsole aus

**WRITE (\*,\*) TRIM(many\_letters)**! Gibt nur "not 30 characters yet" aus

# CHARACTER

## Operatoren:

- // (Konkatenation = Verkettung)
- Vergleichsoperatoren <, >, <=, >=, ==, /= (auf Basis der lexikalischen Ordnung, nachfolgende LZ werden ignoriert)

## Intrinsische Funktionen:

- Intrinsische Funktionen: LEN, TRIM, LENTRIM, ADJUSTL, ADJUSTR

## Beispiel:

```
CHARACTER(len = 12) :: Name, Vorname, Nachname
```

```
Vorname = 'Tom'
```

```
Nachname = "Schulz"
```

```
Name = Vorname // Nachname
```

```
Name = TRIM(Vorname) // ' ' // TRIM(Nachname)
```

```
Name(2:2) = "i"
```

```
! Name = Tom
```

```
! Name = Tom Schulz
```

```
! Name = Tim Schulz
```

# Aufgaben

1. Schreibt ein Programm, das den Nutzer nach Vor- und Nachnamen (max. 30 Zeichen) fragt, und dann folgende Sätze ausgibt:  
"Hallo *Mara Muster*!"  
"*Mara* ist ein schöner Name. :)"

# LOGICAL

## LOGICAL

- Variablen vom Typ LOGICAL können die Werte .TRUE. oder .FALSE. annehmen.
- Operatoren sind .NOT. , .AND. , .OR. , .NEQV. , .EQV.
- Zweimal .NOT. hintereinander ist verboten
- Folgende logische Ausdrücke sind ganz böse und unnötig und sollten nicht verwendet werden:

```
IF (<logexpression>) THEN  
    <logvar>=.TRUE.  
ELSE  
    <logvar>=.FALSE.
```

```
END IF
```

! Viel besser:

```
<logvar> = <logexpression>
```

Zweites Problem

```
IF (<logexp> .EQV. .TRUE.) THEN
```

! Viel besser:

```
IF (<logexp>) THEN
```



# Typvereinbarungen

## Attribute

Jede Variable hat einen Datentypen und kann zusätzlich noch Attribute haben, z. B.

PARAMETER, INTENT(IN), INTENT(OUT), POINTER, OPTIONAL,  
ALLOCATABLE, DIMENSION, INTENT(INOUT), TARGET, SAVE,  
PRIVATE, PUBLIC

Die Syntax für **Typvereinbarungen** ist:

Datentyp `[([KIND =] lwert)] [, Attributliste] :: Variablennamen`

Zwischen dem Datentyp und den `::` kann man Attribute (ein oder mehrere) angeben.

# Typvereinbarungen

## Beispiel (Parameter-Attribut)

Eine Variable mit dem `PARAMETER`-Attribut ist eine Konstante, die zu Beginn des Programms einen festen Wert erhält, welcher nicht mehr geändert werden kann.

```
PROGRAM example_parameter  
  IMPLICIT NONE
```

```
  ! Jetzt kommt eine Variablendeklaration für e mit dem  
  ! PARAMETER-Attribut, weswegen e nicht mehr überschrieben  
  ! werden kann.
```

```
  REAL, PARAMETER :: e = 2.718
```

```
END PROGRAM example_parameter
```

# Aufgaben

- 1 Überarbeitet nun das Programm zur Berechnung des Kreisumfangs vom Beginn der Stunde. Statt als "normale" Variable soll  $\pi$  jetzt als Konstante (PARAMETER) definiert werden.
- 2 **ZUSATZ:** Schreibt ein Programm logical.f95, in welchem ihr zwei LOGICAL-Variablen deklariert und diese mit verschiedenen Wahrheitswerten belegt. Verknüpft die zwei Werte jeweils mit den logischen Operatoren und gebt die Ergebnisse auf dem Terminal aus.
- 3 **ZUSATZ:** Schreibt ein Programm wahrheitswerte.f95 zur "Berechnung" der Wahrheitswerttabellen der logischen Operatoren AND und OR.

# Zusammenfassung

## Einfache Rechnungen

`+, -, *, /, **`

`MOD`

## Vergleiche

`==, / =, <=, >=, >, <`

## Aussagenlogik

`.AND., .OR., .NOT., .EQV., .NEQV.`