

Debuggen und Testen

Sonderübung

Dana Liebscher

PR10

12.11.2023

1. Wiederholung
 - Verzweigungen
 - Schleifen
2. Debuggen und Testen
 - Testen
 - CPU_TIME, MOD und MODULO
 - Aufgabe: Schaltjahre
3. Unterprogramme (Funktionen und Subroutinen)
 - Vorteile von Funktionen
 - Unterprogramme
 - Syntax
 - FUNCTION
 - SUBROUTINE
 - Übungsaufgaben
 - Taschenrechner

Was haben wir in der letzten Übung besprochen?

Verzweigungen

- IF - THEN - ELSE - END IF

```
IF (Bedingung) THEN
    Anweisungsblock
ELSE
    Anweisungsblock
END IF
```

- SELECT CASE:

```
SELECT CASE (Variable)
    CASE (Wertebereich)
        Anweisungsblock

    ...
    CASE DEFAULT
        Anweisungsblock
END SELECT
```

Was haben wir in der letzten Übung besprochen?

Schleifen

- Klassische DO - Schleife:

```
DO
    Anweisungsblock
END DO
```

ACHTUNG: Schleifenabbruch nicht vergessen!

- DO WHILE - Schleife:

```
DO WHILE (Bedingung)
    Anweisungsblock
END DO
```

- Zählschleife:

```
DO k = a, e [, i]
    Anweisungsblock
END DO
```

Warum kompiliert das schon wieder nicht?!

Gerade am Anfang sind Fehlermeldungen manchmal eher verwirrend als hilfreich. Wie kommt man trotzdem zurecht und was sind "typische" Fehler?

- Arbeite die Fehlermeldung von **oben nach unten** ab!
- Kontrolliere an der gekennzeichneten Stelle die Syntax der Verzweigungen/Schleifen (typischer Fehler: "then" bei Verzweigung vergessen)
- Prüfe, ob alle Variablen deklariert sind
- Prüfe, ob du "==" und "=" nicht verwechselt hast
- Bei Zählschleifen: Prüfe, dass du die Zählvariable nicht in der Schleife verändert hast
- Google die Fehlermeldung, frage deine Tutoren oder Kommilitonen

Warum kompiliert das schon wieder nicht?!

Es gibt ein paar "typische" Fehlermeldungen, die mal immer wieder sieht. Hier ein paar Beispiele:

- Error: Unclassifiable statement at (1)
- Error: Function 'FUN' at (1) has no IMPLICIT type
- Program received signal SIGSEGV: Segmentation fault - invalid memory reference.

Testen

Auch wenn das Programm kompiliert hat, macht es noch nicht immer das, was es soll.

- Teste zuerst mit Werten, bei denen du das Ergebnis kennst/ leicht nachrechnen kannst
- Probiere, alle möglichen Fälle bei den Tests abzudecken

Was tun, wenn das Programm nicht das Richtige tut?

- Prüfe, ob du jeder Variable vor ihrer ersten Benutzung einen Wert zugewiesen hast.
- Lasse dir **Zwischenergebnisse** ausgeben!

Was kann man bei Schleifen tun, um nicht von zu vielen Zwischenausgaben überfordert zu werden?

- Ändere die Schleife fürs Testen in eine Zählschleife.
- Füge eine leere READ-Anweisung ein. Der nächste Schleifendurchgang wird erst dann ausgeführt, wenn du "Enter" drückst.

Nützliche Funktionen

CPU_TIME

```
REAL :: start, end
```

```
CALL CPU_TIME(start)
```

```
! do something
```

```
CALL CPU_TIME(end)
```

```
WRITE(*,*) 'Gesamtzeit: ', end - start
```

MOD(a,b)

Wenn a und b beide positiv sind, gibt dieser Befehl den Rest bei der Division von a durch b zurück. Nützlich ist dies z. B. beim Testen von Teilbarkeiten (z.B.

$\text{MOD}(a,4)=0$ bedeutet "Ist a durch 4 teilbar?").

Haben a und b unterschiedliche Vorzeichen, arbeiten die Befehle MOD und MODULO unterschiedlich.

Die gute Nachricht: Oft braucht man nur die Reste positiver Zahlen und es ist daher egal, was ihr verwendet.

$$\text{MOD}(a,b) \rightarrow a - (a/b)*b$$

$$\text{MODULO}(a,b) \rightarrow a - \lfloor \frac{a}{b} \rfloor * b$$

Schachteln oder Verknüpfen von IF- und DO-Anweisungen

Für die Berechnung von Schaltjahren gibt es folgende drei Regeln:

- Ist die Jahreszahl durch 4 teilbar, dann ist das Jahr ein Schaltjahr.
- Ist die Jahreszahl jedoch durch 4 und 100 teilbar, dann ist das Jahr kein Schaltjahr.
- Ist die Jahreszahl durch 4, 100 und 400 teilbar, dann ist das Jahr ein Schaltjahr.

Schreibe ein Programm, welches alle Schaltjahre zwischen zwei eingelesenen Jahren ausgibt. Finde geeignete Testwerte, um alle vier Fälle abzudecken.

Zusatz: Nutze die CPU_TIME Funktion um bei großen Intervallen die Programmeffizienz (-laufzeit) zu messen.

Hinweis: $\text{MOD}(a,b)$ liefert den Rest von a bei der Teilung durch b .

Vorteile von Funktionen

- Auslagern von Code \Rightarrow übersichtlicher
- Vermeiden von Copy-und-Paste-Programmierung \Rightarrow besserer Stil
- Ihr könnt jederzeit auf eure Funktion zugreifen
- Komplexere Programme möglich

Unterprogramme

- implementieren Teilalgorithmen
- UP in Fortran sind entweder eine **Funktion** oder eine **Subroutine**
- Jedes UP besitzt **Kopf** und **Rumpf**

Funktionen

- liefern ein Ergebnis, d. h. besitzen **genau einen** Rückgabewert
- Funktionsaufrufe stehen in Ausdrücken

Subroutinen

- liefern **kein** Ergebnis
- werden mittels CALL-Anweisung aufgerufen

SYNTAX

```
PROGRAM up  
  IMPLICIT NONE  
  ...
```

! Hauptprogramm

```
CONTAINS
```

```
  FUNCTION myfun (a, b, c [, ...])  
    INTEGER :: a, b  
    LOGICAL :: c  
    INTEGER :: myfun  
    ...
```

· formale Argumente
! Variablendeklaration

```
  END FUNCTION
```

```
  SUBROUTINE mysub (a, b, c [, ...])  
    INTEGER :: a, b  
    LOGICAL :: c  
    ...
```

! Variablendeklaration

```
  END SUBROUTINE
```

```
END PROGRAM
```

FUNCTION

Beispiel

```
FUNCTION area (d)
  REAL, PARAMETER :: pi = 3.14159
  REAL :: d
  REAL :: area

  area = 0.25 * pi * d ** 2
END FUNCTION
```

· formale Argumente

Aufruf (in einem Ausdruck)

```
res_area = area(5) + area(8) * 2
oder
WRITE(*,*) 'Die Flaeche betraegt ', area(diameter)
```

· aktuelle Argumente

SUBROUTINE

Beispiel

```
SUBROUTINE print (d)                                · formale Argumente
    REAL :: d

    WRITE(*,*) 'Der Durchmesser betraegt ', d, ' cm.'
END SUBROUTINE
```

Aufruf (mittels CALL-Anweisung)

```
CALL print (2 * r)                                  · aktuelle Argumente
```


Übungsaufgabe

Unser Taschenrechner soll noch mehr Funktionalität erhalten.
Schreibe Unterprogramme, die die folgenden Berechnungen durchführen:

- Berechnung der Fläche eines Kreises
- **Zusatz:** Berechnung des Volumens und der Oberfläche eines Quaders (zusammen in einem Unterprogramm)
- **Zusatz:** Kommentierte Ausgabe von Fläche und Volumen (des Kreises bzw. Quadrats)