

Unterprogramme

Sonderübung

Luise Zieger & Nadine Schärmann

PROG

11. Dezember 2019

Inhalt

1. Wiederholung
 - Syntax
 - FUNCTION
 - SUBROUTINE
2. Übungsaufgaben zur Wiederholung
3. Rekursion
4. Übungsaufgabe
5. Werte- und Referenzparameter
6. Attribute für formale Argumente
 - INTENT(IN)
 - INTENT(OUT)
 - INTENT(INOUT)
7. Zusammenfassung

Wiederholung

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
```

Übungsaufgaben

Unser Taschenrechner soll noch mehr Funktionalität erhalten.

- Schreibe eine Funktion, die die n -te Dreieckszahl $(1+2+\dots+n)$ berechnet
- **Zusatz:** Schreibe eine Funktion, die den Binomialkoeffizienten $\binom{n}{m}$ berechnet

Rekursive Unterprogramme

Ein rekursives UP ruft sich selbst auf (direkt oder indirekt, d. h. über andere UPe). Ein **RECURSIVE** - Attribut im UP-Kopf ist notwendig. Funktionen brauchen zusätzlich noch eine **RESULT (res)** -Klausel.

Beispiel

```
RECURSIVE FUNCTION fibo (n) RESULT (res)
  INTEGER :: n
  INTEGER :: res

  IF (n == 0 .OR. n == 1) THEN
    res = 1
  ELSE
    res = fibo (n-1) + fibo (n-2)
  END IF
END FUNCTION
```

ACHTUNG: Abbruchbedingung nicht vergessen!

Übungsaufgabe

Aufgabe

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

- Rekursive Berechnung der Fakultät einer ganzen Zahl ($n!$)
- Rekursive Berechnung der zu einer ganzen Zahl gehörigen Dreieckszahl ($1 + \dots + n$)
- **Zusatz** Rekursive Berechnung der Binomialkoeffizienten unter Ausnutzung von $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$

Werte- und Referenzparameter

Werteparameter (= value parameter)

- Ergebniswert des aktuellen-Argument-Ausdrucks wird dem formalen Argument zugewiesen
- verhält sich anschließend wie lokale Variable im UP

Referenzparameter (= reference parameter)

- f. A. wird mit a. A. (Variable) assoziiert
- d. h. wird der Wert des f. A. im UP verändert, so ändert sich auch der Wert des a. A.

WICHTIG: In Fortran gibt es nur Referenzparameter.

Das heißt, wenn ihr eine Variable in einem Unterprogramm verändert, ändert sich ihr Wert dauerhaft (d.h. auch nach Beendigung des UP ist der Wert anders).

Attribute für formale Argumente

INTENT(IN)

- das assoziierte a. A. muss einen Wert haben
- Wert des a. A. darf im UP nicht verändert werden
- garantiert, dass die Variable (a. A.) nach dem UP-Aufruf unverändert ist
- so dürfen auch Ausdrücke übergeben werden

Attribute für formale Argumente

INTENT(OUT)

- f. A. muss zu Beginn des UPs (noch) keinen def. Zustand haben
- daher ist lesender Zugriff auf das f. A. zu Beginn des UPs unzulässig
- f. A. muss am Ende des UPs einen definierten Zustand haben
- garantiert, dass die übergebene Variable (a. A.) nach dem UP einen def. Zustand besitzt
- Ausdrücke dürfen nicht übergeben werden

Attribute für formale Argumente

INTENT(INOUT)

- Alles erlaubt (lesen und schreiben aus der/ in die Variable)

Übungsaufgabe

Aufgabe

- 1 Überarbeite den Taschenrechner folgendermaßen: Füge den formalen Argumenten der Unterprogramme die Attribute `INTENT(IN)`, `INTENT(OUT)` bzw. `INTENT(INOUT)` hinzu.

Collatz-Vermutung

Betrachte folgende Funktion:

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

$$f(n) = \begin{cases} 3n + 1 & n \text{ ungerade} \\ n/2 & n \text{ gerade} \end{cases}$$

Die Collatz-Vermutung besagt, dass unabhängig von der Zahl mit der man anfängt, irgendwann 1 herauskommt. Zum Beispiel ergibt sich beim Start mit 19 folgende Folge:

19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Schreibe ein Programm, das eine Zahl einliest und für diese in einer Funktion berechnet, wie viele Schritte benötigt werden, bis sich 1 ergibt.

Zusatz: Berechne, bei welcher Zahl zwischen 1 und 100 sich die längste Kette ergibt.

Zusammenfassung

- **Funktionen**

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

- **Subroutinen**

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

- **Rekursive UP** rufen sich selbst (direkt oder indirekt) auf

- In Fortran gibt es nur **Referenzparameter** (d.h. f. A. wird mit a. A. assoziiert = Alias)

- Attribute für formale Argumente sind:

INTENT(IN), **INTENT(OUT)** und **INTENT(INOUT)**