

Unterprogramme

Sonderübung

Dana Liebscher

PR10

19.11.2024

Inhalt

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

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

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)

- formales Argument wird mit aktuellem Argument (Variable) assoziiert
- d. h. wird der Wert des formalen Arguments im UP verändert, so ändert sich auch der Wert des aktuellen Arguments

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)

- formales Argument wird mit aktuellem Argument assoziiert (Referenzparameter)
- das assoziierte aktuelle Argument muss einen Wert haben
- Wert des aktuellen Arguments darf im UP nicht verändert werden
- garantiert, dass die Variable (aktuelles Argument) nach dem UP-Aufruf unverändert ist
- so dürfen auch Ausdrücke übergeben werden

Attribute für formale Argumente

INTENT(OUT)

- formales Argument wird mit aktuellem Argument assoziiert (Referenzparameter)
- formales Argument muss zu Beginn des UPs (noch) keinen def. Zustand haben
- daher ist lesender Zugriff auf das formale Argument zu Beginn des UPs unzulässig
- formales Argument muss am Ende des UPs einen definierten Zustand haben
- garantiert, dass die übergebene Variable (aktuelles Argument) nach dem UP einen def. Zustand besitzt
- Ausdrücke dürfen nicht übergeben werden
- Return-Variablen in Funktionen dürfen **keine** INTENT-Attribute bekommen.

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.

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) THEN
    res = 0
  ELSE IF (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 $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$

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. formales Argument wird mit aktuellem Argument assoziiert = Alias)

- Attribute für formale Argumente sind:

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

Was ist Modularisierung?

- Idee: komplexes System nach dem Baukastenprinzip aus Einzelbausteinen zusammensetzen
- findet im alltäglichen Leben Anwendung, aber auch in vielen Bereichen der Informatik (insbesondere bei der Entwicklung von Programmen)
- UPe entsprechen einfacher Umsetzungen des Modularisierungsprinzips
- UPe als Bausteine bei der Entwicklung von Programmen verwendbar, falls die UPe als eigenständige Verarbeitungseinheiten konzipiert sind

Was ist Modularisierung?

Grundlegende Struktur eines Moduls

```
MODULE < Modulname >  
    IMPLICIT NONE  
    PRIVATE          ! anything that is not declared PUBLIC is PRIVATE  
    PUBLIC :: < Exportliste >  
    ...  
    CONTAINS  
        < UPe >  
END MODULE [< Modulname >]  
  
PROGRAM < Programmname >  
    USE < Modulname >  
    IMPLICIT NONE  
    ...  
END PROGRAM [< Programmname >]
```

Übungsaufgabe

Erstelle ein Modul mit allen zu dem Taschenrechner gehörigen Funktionen.