Programmsteuerung

Mittel zur Programmverzweigung:

GOTO–Anweisung:

- Unbedingtes GOTO:
  `GOTO L` (L=Labelnummer)
- Bedingtes GOTO (Finger weg!):
  `GOTO(L1,L2,...) I`
  (L1,L2,...=Labelnummer, I: Integervariable, die angibt, welche Labelnummer verwendet wird)

IF–Anweisung:

- Logisches IF:
  `IF (<logischer Ausdruck>) <Statement>`
  (Statement kann jede ausführbare Anweisung sein außer `IF, DO, ELSE, ELSE IF, ENDIF, END`), z.B.:

```
IF(A.GT.10) B=3.0
```

- Arithmetisches IF (Finger weg!):
  `IF(<arithmetischer Ausdruck>) L1,L2,L3`

Block–IF–Anweisung:

- einfachste Form:

```
IF(<logischer Ausdruck>) THEN
   ⋮
ENDIF
```

- ELSE–Anweisung:

```
IF(<logischer Ausdruck>) THEN
   ⋮
ELSE
   ⋮
ENDIF
```

- Verschachtelung mit ELSE oder ELSE IF, z.B.:

```
IF(<logischer Ausdruck>) THEN
   ⋮
ELSE
   IF(<logischer Ausdruck>) THEN
      ⋮
   ELSE
      ⋮
   ENDIF
ENDIF
```

- nur ineinander schachteln!
- Einrückungen für Übersichtlichkeit
- Beispiel:

```
IF(A.GT.10) THEN
   X = 3.*Y + 2.
ELSE
   IF(B.GT.0.) THEN
      Z = 3.5
      X = 2.0
   ELSE
      X = 3.0
   ENDIF
ENDIF
```

oder (ELSE IF–Konstruktion)

```
IF(A.GT.10) THEN
   X = 3.*Y + 2.
ELSE IF(B.GT.0.) THEN
      Z = 3.5
      X = 2.0
   ELSE
      X = 3.0
ENDIF
```

DO–Schleifen

```
DO L I=M1,M2,M3
```

(L=Labelnummer, I=Schleifenindex, M1=Startwert, M2=Endwert, M3=Schrittweite (Default=1))

- `I,M1,M2,M3` sollen/dürfen in der Schleife nicht verändert werden.
- Letzte Anweisung muß ausführbar sein, d.h. nicht: DO, GOTO, PAUSE, RETURN, arithm. IF, Block–IF, ELSE IF, ELSE, ENDIF, END
- In der Schleife darf kein RETURN auftreten.
- **Empfehlung 1:** `CONTINUE` als letzte Anweisung einer Schleife
- **Empfehlung 2:** INTEGER–Variable als Schleifenindex (reelle Zahlen möglich, aber *loop*–Länge hängt von interner Rechengenauigkeit ab!)

Beispiel:

```
      DO 100 I=1,15
         J = 3 + I
             ⋮
100   CONTINUE
```

Alternative Form der DO-Schleife

Kein Standard in FORTRAN77, erst in FORTRAN90 Standard!

```
DO I=M1,M2,M3
   ⋮
END DO
```

Beispiel:

```
DO I=1,15
   J = 3 + I
      ⋮
END DO
```

Nützliche Iterationen

In anderen Programmiersprachen wie PASCAL gibt es nützliche Konstrukte wie DO–WHILE– und DO–UNTIL–Iterationen, die in FORTRAN77 nicht zur Verfügung stehen, daher:

- **DO–WHILE in FORTRAN77:**

```
      N = 1
100   IF(N.LT.1000) THEN
         N = 2 * N
         J = J + 1
         GOTO 100
      ENDIF
```

- **DO–UNTIL in FORTRAN77:**

```
      N = 256
100   CONTINUE
         N = N / 2
         J = J + 1
      IF(N.NE.1) GOTO 100
```

- Initialisierung nicht vergessen ($\Rightarrow$ Endlos–Schleife!)

Kommentare

Kommentare beginnen in der ersten Spalte:

```
C       Dies ist ein Kommentar
```

Beim Kommentieren sollte man folgendes beachten:

- Übersichtliche, ausgewogene und strukturierte Kommentare
- ... im *header* eines Programmsegments:
  - Zweck der Routine
  - Hinweis auf Methode/Algorithmus
  - Ein– und Ausgabeparameter
  - Besonderheiten
  - Autoren bzw. Entwicklungs'geschichte', Version, Datum
- ... im Rumpfteil
  - Gliederung logisch gekoppelter Abschnitte
  - algorithmische Hinweise
  - Unterprogrammaufrufe, komplizierte Funktionen
  - nicht Überkommentieren!

```
      PROGRAM TEST
      A=0.00001
      B=1.
      C=1.
      N=1
10    CONTINUE
      F=B+C
      E=ABS(F-B)
      WRITE(*,*) 'N = ',N,'    F= ',F
      N=N+1
      B=F
      C=C/FLOAT(N)
      IF (E.GT.A) GOTO 10
      END
```

Indizierte Variable (Felder)

Vektoren, Matrizen o. ä. können mit Feldern (*array*) dargestellt werden.

- Deklaration:

```
REAL A
DIMENSION A(N)
```

- Besser:

```
REAL A(N)
```

(Indexbereich von 1 bis N)

oder

```
REAL A(M:N)
```

(M. . .N: Bereich 'erlaubter' Indices)

- Index–Bereich darf negativ sein.
- bis zu 7 Indices erlaubt
  (**Empfehlung:** nicht mehr als 3 Indices)

```
REAL A(M,N)
```

(1. Index von 1 bis M, 2. Index von 1 bis N)

- Darstellung im Speicher: eindimensional, seriell
- Der am weitesten links stehende Index läuft am schnellsten. (bei C gerade umgekehrt!)

- Indices dürfen aus einfachen arithmetischen Berechnungen (`*,+,-`) bestehen.
- Überschreiten der Indices führt normalerweise zu keiner Fehlermeldung! $\Rightarrow$ Kontrolle durch Programm(ierer)In! (aber: sog. *array–bound–checking* häufig als Compiler–Option vorhanden)
- keine Vektornotation (z.B. a=0.)

Beispiele:

1.
```
REAL A(6),B(-3:5),C(-1:1,2:5)
```

2.
```
PARAMETER(NMAX=1000)
REAL A(NMAX),B(NMAX,NMAX+1)
```

3.
```
X = A(5*I+3)
```

Programmsegmente

Programmsegmente können nacheinander in einem File stehen oder auf mehrere Files verteilt werden, die vom Compiler einzeln bearbeitet werden.

1. **HAUPTPROGRAMM**

```
PROGRAM name
  ⋮
END
```

2. **UNTERPROGRAMM**

```
SUBROUTINE name (<Parameterliste>)
<Deklarationsteil>
<Anweisungsteil>
  ⋮
RETURN
END
```

- Parameterliste darf leer sein.
- Beispiel:

```
SUBROUTINE UNTER(X,Y,Z)
```

oder

```
SUBROUTINE SUB1
```

- Aufruf des Unterprogramms:

```
CALL <Unterprogrammname>(<Parameterliste>)
```

z.B.

```
CALL UNTER(A,B,C)
```

oder

```
CALL SUB1
```

- Unterprogramm darf weitere Unterprogramme aufrufen.
- keine rekursiven Aufrufe (in FORTRAN90 gibt es die Möglichkeit rekursive Funktionen zu definieren)
- In verschiedenen Segmenten dürfen gleiche Namen für unterschiedliche Variable benutzt werden (lokale Variable).
- Weiterreichen von Variablen erfolgt entweder durch Parameterliste oder über COMMON–Block.

- Beispiel:

```
      ⋮
      A = 5.5
      B = 3.2
      CALL SUB(A,B,C)
      ⋮
      END
C*****************************
      SUBROUTINE SUB(X,Y,Z)
      REAL X,Y,Z
      Z = X * Y
      END
```

- A,B,C sind **aktuelle Parameter**.
- X,Y,Z sind **formale Parameter**, d.h. sie belegen keinen Speicherplatz.

Bemerkungen zu den Übergabeparametern – 1

- Variable werden im *call–by–reference*–Verfahren übergeben, d.h. es wird nur die Speicheradresse, nicht aber der Variablenwert weitergegeben (sog. *call–by–value*).
- Übergeben werden dürfen Variable und Konstanten; letztere dürfen aber KEINESFALLS im Unterprogramm verändert werden (wird nicht vom Compiler überprüft!).
- Auch Zahlen dürfen direkt in der Parameterliste übergeben werden (⇒ Konstante).
- FORTRAN überprüft nicht, ob die übergebenen Variablen den im Unterprogramm deklarierten und erwarteten Variablentyp entsprechen(feature oder bug?!).

  ⇒ ProgrammiererIn muß selbst Konsistenz der Übergabeliste sicherstellen!

  (häufigste ernsthafte Fehlerquelle, kryptische Ergebnisse!)
- **Empfehlung:** keine Zahlen in CALL–Aufruf, explizites Prüfen der Variablenliste auf Konsistenz

3. FUNKTION

```
<Funktionstyp> FUNCTION name(<Parameterliste>)
   ⋮
name = ...
END
```

- Parameterliste darf nicht leer sein (*dummy*–Argument darf aber übergeben werden).
- Parameter wird wie bei SUBROUTINE übergeben (call–by–reference)
- Funktionstypen: REAL, INTEGER, LOGICAL, DOUBLE PRECISION, CHARACTER, COMPLEX (ansonsten gilt die implizite Typvereinbarung)
- Dem Funktionsnamen **muß** innerhalb der Funktion ein Wert zugewiesen werden.
- Benutzerdefinierte Funktionen müssen im rufenden Programm deklariert werden.
- Beispiel:

```
DOUBLE PRECISION FUNCTION F1(X)
DOUBLE PRECISION X
F1 = X**2
R E T U R N
END
```

vordefinierte (intrinsische) Funktionen

- Wurzel: `SQRT(X)`
- Exponentialfunktion: `EXP(X)`
- Betrag: `ABS(X)`
- Vorzeichen: `SIGN(X,Y) = sgn(Y) * |X|`
- Maximum/Minimum: `MAX(X,Y)`, `MIN(X,Y)`
- ganzzahliger Rest bei Division: `MOD(X,Y)`
- natürlicher Logarithmus: `LOG(X)`
- dekadischer Logarithmus: `LOG10(X)`
- trigonometrische Funktionen, z.B. `SIN(X)`
- zugehörige Umkehrfunktionen, z.B. `ACOS(X)` (Besonderheit: auch 'quadrantengerechter' Arcustangens: `ATAN2(Y,X)`)
- Hyperbelfunktionen, z.B. `SINH(X)` (aber nicht die zugehörigen Umkehrfunktionen!)
- Typkonversion: `REAL(I)`, `DBLE(I)`, `INT(X)`, `NINT(X)`,...
- ...
- + maschinen–abhängige Funktionen (z.B. CPU–Zeit, Datum, weitere Funktionen etc.)

4. COMMON–BLOCK

```
COMMON /<COMMON-Block-Name>/ <Variablenliste>
```

- COMMON–Blöcke gestatten gemeinsamen Zugriff unterschiedlicher Programmsegmente auf gleiche Speicherbereiche ($\Rightarrow$ eine weitere Variante Variablen 'auszutauschen').
- COMMON–Block muß im Deklarationsteil eines Segments stehen.
- COMMON–Block ist ein eigenständiges Segment, das im Kernspeicher hinter dem Segment steht, in dem er das erste Mal vorkommt.
- FORTRAN überprüft nicht die Konsistenz der COMMON–Blöcke in verschiedenen Segmenten.

ProgrammiererIn muß selbst Konsistenz der COMMON–Blöcke, d.h. der in ihnen definierten Variablen, sicherstellen!

("schöne" Quelle für unauffindbare Fehler...!)

- **Beispiel:**

```
      PROGRAM PROG1
      DOUBLE PRECISION X
      COMMON /PARA/ X
      CALL SUB1
      WRITE(6,*) X
      END
C*************************************
      SUBROUTINE SUB1
      DOUBLE PRECISION X
      COMMON /PARA/ X
      X=4.D0
      END
```

- **1. Empfehlung:** Bei Änderungen an einem COMMON–Block **SOFORT** alle anderen Stellen, an denen er auftritt, ebenfalls aktualisieren.
- **2. Empfehlung:** **NIEMALS** unterschiedliche Variablentypen verwenden oder Reihenfolge der COMMON–Block–Variablen ändern.
- **3. Empfehlung:** Verwendung von INCLUDE–Files

INCLUDE–Files

- INCLUDE–Files können an beliebiger Stelle im Programm eingefügt werden (allerdings Nicht–Standard–FORTRAN77).
- INCLUDE–Files enthalten typischerweise alle relevanten Deklarationen und COMMON–Blöcke, z.B.

File prog1.f

```
      PROGRAM PROG1
      INCLUDE 'prog1.inc'
      CALL SUB1
      WRITE(6,*) X
      END
C*************************************************
      SUBROUTINE SUB1
      INCLUDE 'prog1.inc'
      X=4.D0
      END
```

File prog1.inc

```
      DOUBLE PRECISION X
      COMMON /PARA/ x
```