FORTRAN 90

Rückwärtskompatibel: alte FORTRAN 77-, FORTRAN IV-Programme können weiterbenutzt werden.

Neue Konzepte:

- Freies Format: alternatives Format für Programme
- Abgeleitete Typen (Strukturen)
- neue Funktionen, z.B. für Feldoperationen
- Vergleichsoperatoren: alternative Schreibweisen
- alternative Schleifen und Kontrollstrukturen
- andere Formen der Parameterübergaben an Unterprogramme
- rekursive Funktionsaufrufe
- dynamische Speicherverwaltung

freies FORTRAN 90-Format

- Programmzeilen können bis zu 132 Zeichen lang sein
- Die Anweisungen können an beliebiger Stelle stehen
- bis zu 39 Fortsetzungszeilen sind möglich. Dafür setzt man ein **&** an das Ende der vorhergehenden Zeile. Evt. spezielle Vorgehensweise für Zeilenübergreifende CHARACTER-Konstanten notwendig.
- Mehrere Anweisungen können auf einer Zeile stehen. Trennung durch **;** . (Allerdings sparsame Anwendung zu empfehlen).
- Label stehen wie bisher vor den Anweisungen – keine Beschränkung auf die ersten 5 Spalten.
- Kommentare werden durch **!** eingeleitet (kann auch hinter Anweisungen stehen). Der Rest der Zeile ist Kommentar.
- Variablennamen dürfen bis 31 Zeichen lang sein und neben Zahlen und Buchstaben auch _ (Underscore) enthalten
- Quelltext im freien Format wird durch die Extension `f90` erkannt.

Typ-Deklarationen

Neue Schreibweise für die Typdefinition, z.B.:

```
INTEGER          :: I
REAL             :: A, B
```

Weitere Spezifikationen:

```
REAL(KIND=LONG)     :: C
CHARACTER(LEN=20)   :: NAME
REAL, DIMENSION(10):: ARRAY
```

Kurzfassungen:

```
REAL(LONG)       :: C
CHARACTER(20)    :: NAME
```

Strukturen

Definition eines abgeleiteten Typs (Struktur):

```
TYPE PERSON
  CHARACTER(LEN=10) :: NAME, VORNAME
  REAL              :: ALTER
  INTEGER           :: ID
END TYPE PERSON
```

Definition einer Struktur:

```
TYPE(PERSON)   :: ER, SIE, ES
```

Wertzuweisungen an Strukturen

Ansprechen einzelner Komponenten durch Name der Struktur und Name der einzelnen Komponente mit dem Komponenten-Selektor % , z.B. ER%NAME.

```
ER%NAME = 'FRITZ'
ER%ID   = ER%ID + 9
DIFF    = ER%ALTER - SIE%ALTER
```

Ausdrücke der Form ER + SIE sind undefiniert, können aber definiert werden.
Konstanten:

```
PERSON('Franz', 'Hans', 37, 4509889680)
```

Zuordnung:

```
ER = PERSON('Franz', 'Hans', 37, 4509889680)
```

Definitionen von Strukturen können auch Strukturen verwenden:

```
TYPE PUNKT
  REAL  :: X, Y
END TYPE PUNKT
TYPE DREIECK
  TYPE(PUNKT)  :: A, B, C
END TYPE DREIECK
TYPE(DREIECK) :: GROSS, KLEIN
   ...
GROSS%A = PUNKT( 0.7,0.4)
   ...
```

Neue Funktionen

Zahlenmodelle etc.

`EPSILON(X)`: Zahl, die für Variablen vom Typ `X` klein gegen 1 ist.

`PRECISION(X)`: maximale Anzahl signifikanter Dezimalstellen für Variablen vom Typ `X`

`HUGE(X)`: Größte darstellbare Zahl für Variablen vom Typ `X`

`TINY(X)`: Kleinste (positive) darstellbare Zahl für Variablen vom Typ `X` .

...

Bitoperationen auf Integer-Zahlen

`IOR(X,Y)`: Bitweise `.OR.`-Verknüpfung der INTEGER-Zahlen X und Y.

...

Bearbeiten von Zeichenketten

`ADJUSTL(STRING)`: Entfernen führender Leerstellen (linksbündig)

`ADJUSTR(STRING)`: rechtsbündig

`LEN_TRIM(STRING)`: Länge von `STRING` ohne Leerzeichen am Ende.

...

Matrixoperationen

Die üblichen Verknüpfungen funktionieren auch mit Feldern.

- Die Operationen werden elementweise abgearbeitet.
- Die Felder müssen die gleichen Dimensionierungen haben.
- Man kann auch Skalare und Felder kombinieren.

`FELD2 = FELD1`: Die Elemente von `FELD2` werden denen von FELD1 gleichgesetzt

`FELD1 * FELD2`: FELD1 wird elementweise mit FELD2 multipliziert.

`5. * FELD1`: Alle Elemente von FELD1 werden mit 5 multipliziert.

`FELD1 .EQ. FELD2`: Ergebnis ist ein Feld des Typs LOGICAL mit den Werten .TRUE./.FALSE.

Feld-Funktionen

`SUM(FELD)`: Summe aller Feldelemente

`PRODUCT(FELD)`: Produkt aller Feldelemente

`MAXVAL(FELD), MINVAL(FELD)`: Maximales/minimales Feldelements

`MAXLOC(FELD), MINLOC(FELD)`: Indices des maximales/minimales Feldelements

...

Bei diesen Funktionen können Masken angewendet werden. Das sind logische Felder, die angeben, ob ein Element des numerischen Feldes bei der Operation berücksichtigt werden soll, oder nicht.

Matrixmultiplikation u.ä.

`DOT_PRODUCT(A,B)`: Skalarprodukt der Vektoren A und B.

`MATMUL(A,B)`: Matrixmultiplikation der Matrizen A und B. **Eine** Matrix darf ein Vektor sein. Die Dimensionierungen müssen die üblichen Regeln der Matrixmultiplikation entsprechen.

`TRANSPOSE(MATRIX)`: Liefert die Transponierte einer MATRIX.

...

Datum, Zeit

`DATE_AND_TIME (DATE, TIME, ZONE, VALUES)`: Datum und Uhrzeit

`CPU_TIME (TIME)`: Gibt in `TIME` (Typ REAL) die verbrauchte CPU-Zeit in Sekunden zurück (Achtung: FORTRAN 95).

Vergleichsoperatoren

Alternative Schreibweise der Vergleichsoperatoren:

`.LT.`	`<`	kleiner
`.LE.`	`<=`	kleiner gleich
`.GT.`	`>`	größer
`.GE.`	`>=`	größer gleich
`.EQ.`	`==`	gleich
`.NE.`	`/=`	nicht gleich

CASE-Kontrollstruktur

```
SELECT CASE (Case-Ausdruck)
  CASE (Case-Auswahl)
  CASE DEFAULT
END SELECT
```

Case-Ausdruck: LOGICAL-, INTEGER- oder CHARACTER-Ausdruck

Case-Auswahl: (Liste von) Werten oder Wertebereich

`CASE DEFAULT`: Behandelt die verbleibenden Fälle

Beispiel:

```
SELECT CASE (N)
  CASE (:-1)
    N_SIGN = -1
  CASE (0)
    N_SIGN = 0
  CASE (1:)
    N_SIGN = 1
END SELECT
```

DO-Schleifen in FORTRAN 90

Standardschleife:

```
DO I=M1,M2,M3
   ...
END DO
```

WHILE-Schleife:

```
DO WHILE (logischer Ausdruck)
   ...
   ...
END DO
```

Endlos-Schleife

```
DO
   ...
   ...
END DO
```

Schleifenkontrolle:

`EXIT`: Beendet die Schleife

`CYCLE`: Beendet den aktuellen Schleifendurchlauf

Parameterübergabe an Unterprogramme

INTENT- und OPTIONAL-Angaben:

`INTENT(IN)`: Eingabeparameter, darf nicht verändert werden. Konstante oder Variable.

`INTENT(OUT)`: Ausgabeparameter, bei beginn undefiniert. Variable

`INTENT(INOUT)`: Ein- und Ausgabeparameter. Variable

`OPTIONAL`: Parameter ist optional (benötigt ein Interface

Beispiel:

```
SUBROUTINE INTEGRAL (SUMME, UNTEN, OBEN, N, H)

REAL         :: SUMME, UNTEN, OBEN, H
INTEGER      :: NMAX
INTENT(OUT):: SUMME
INTENT(IN)  :: UNTEN, OBEN, NMAX
OPTIONAL     :: N, H

IF (.NOT.PRESENT(N)) THEN
  ...
```

`PRESENT(X)`: logische Funktion. Testet, ob optionale Parameter übergeben wurden.

Parameterübergabe mit Keywords:

```
INTERFACE
  SUBROUTINE INTEGRAL (SUM, LOW, UP, NSTEP, &
                       STEP)

  REAL         :: SUM, LOW, UP, STEP
  INTEGER      :: NSTEP
  INTENT(OUT):: SUM
  INTENT(IN)  :: LOW, UP, STEP
  OPTIONAL     :: NTEP, STEP
  END SUBROUTINE
END INTERFACE

CALL INTEGRAL (SUMME, UNTEN, OBEN, STEP=0.1)
```

(Explizites) Interface:

- Wird im rufenden Programm benötigt falls optionale Argumente, Übergabe per Keyword, u.a.m. verwendet werden,
- enthält den Kopf von Unterprogramm/Funktion mit Formalparametern,
- kann auch zur Definition von "Gattungsnamen" verwendet werden

Zusammenfassung einer REAL-, INTEGER- und DOUBLE-PRECISION-Version zu einem Gattungsnamen:

```
INTERFACE AUSTAUSCH
  SUBROUTINE IAUSTAUSCH (A, B)
    INTEGER :: A, B
  END SUBROUTINE
  SUBROUTINE AUSTAUSCH (A, B)
    REAL :: A, B
  END SUBROUTINE
  SUBROUTINE DAUSTAUSCH (A, B)
    REAL(KIND=DOUBLE) :: A, B
  END SUBROUTINE
END INTERFACE
```

Interfaces u.a. können in Modulen zur Verfügung gestellt werden:

```
MODULE TAUSCHEN
INTERFACE AUSTAUSCH
  SUBROUTINE IAUSTAUSCH (A, B)
    ...
    ...
END INTERFACE
END MODULE
```

Bekanntmachen in einer Programmeinheit: `USE TAUSCHEN`

Rekursive Aufrufe

SUBROUTINE:

```
RECURSIVE SUBROUTINE SUB (I)
INTEGER I
  ...
CALL SUB (I)
  ...
END
```

FUNCTION:

```
RECURSIVE FUNCTION FAKULTAET (N) RESULT(FAK)
INTEGER   :: N, FAK

IF ( N==0 ) THEN
  FAK = 1
ELSE
  FAK = FAKULTAET (N-1)*N
END IF
END
```

Dynamische Felder

FORTRAN 90 kann Felder dynamisch verwalten. Die Felder müssen als ALLOCATABLE definiert sein. Mit ALLOCATE werden die Felder mit den aktuellen Grenzen angelegt und mit DEALLOCATE wieder freigegeben.

```
REAL, ALLOCATABLE :: MATRIX (:,:)
INTEGER :: STATUS
  ...
ALLOCATE (MATRIX (5,10), STAT=STATUS)
! STATUS = 0 ==> kein Fehler
  ...
DEALLOCATE (MATRIX)
```

Der Zustand des Feldes (allocated/nicht allocated) kann mit `ALLOCATED (MATRIX)` abgefragt werden.

Pointer

- Pointer enthalten als Wert die Addresse einer Variablen.
- Wichtigste Anwendung: anlegen von dynamisch verwalteten Listen
- Deklaration mit dem Attribut `POINTER`
- Ziele müssen mit dem `TARGET`-Attribut angelegt werden
- Zuweisung eines Zieles mit `=>`
- Alternativ dynamische Erzeugung von Pointer-Zielen mit `ALLOCATE`

```
INTEGER, POINTER :: PTR1, PTR2
INTEGER, TARGET  :: ZIEL
  ...
ZIEL = 123
PTR1 => ZIEL
  ...
ALLOCATE (PTR2)
PTR2 = 123
  ...
WRITE (*,*) ZIEL, PTR1, PTR2
```

Typische Anwendung:

```
TYPE PERSON
  CHARACTER(LEN=10)    :: NAME, VORNAME
  REAL                 :: ALTER
  INTEGER              :: ID
  TYPE(PERSON),POINTER:: NAECHSTE
END TYPE PERSON
```

`NULLIFY (POINTER)` löscht das Zeigerziel, dieser Pointer und alle anderen, die auf das Zeigerziel zeigen, erhalten den Status undefiniert.

Mit `ASSOCIATED (POINTER)` kann man den Zeigerstatus abfragen.