Fortran 90

Entwicklungsziele für Fortran 8x

- Aufwärtskompatibilität zu FORTRAN 77
- Behebung wesentlicher Mängel von FORTRAN 77
- Modernisierung der Sprache
- Besondere Eignung für wissenschaftlich-technische Anwendungen
- Gewährleistung der Portabilität
- Sprachumfang soll nicht zu komplex werden

- FORTRAN 77 ist in Fortran 90 enthalten

Erweiterter Zeichensatz

- Unterstrich (_) in Namen

- Kleinbuchstaben dürfen verwendet werden

- Neue Sonderzeichen:
  ! " % & ; < > ?

Beispiel für Name: TABULATOR_POSITION

- Namen können max. aus 31 Zeichen bestehen

Formatgebundene Quellform

- ! leitet Kommentar ein

- ; trennt Anweisungen in einer Zeile

Formatfreie Quellform

- Eine Zeile enthält zwischen 0 und 132 Zeichen

- ! leitet Kommentar ein

- ; trennt Anweisungen

- & als letztes nichtleeres Zeichen in einer Zeile kündigt an, daß die Anweisung in der nächsten Zeile fortgesetzt wird

- ZWISCHENRAUM ist signifikant

Nur eine Quellform je Programmeinheit erlaubt

Typ-Deklaration

Typ [[, Liste der Attribute] ::] Liste der Deklarationen

INTEGER
REAL
DOUBLE PRECISION
COMPLEX
CHARACTER
LOGICAL
TYPE

Beispiele:

```
REAL :: A, B, C
INTEGER (KIND = 2) :: I_KURZ
INTEGER (KIND = 4) :: I_LANG
REAL (KIND = 4) :: R_EINFACH_GENAU
REAL (KIND = 8) :: R_DOPPELT_GENAU
COMPLEX (KIND = 4) :: C1
COMPLEX (KIND = 8) :: C2
CHARACTER (LEN = 10, KIND = 1)  :: TEXT, AN
CHARACTER (LEN = 20) :: FORTRAN_TEXT
LOGICAL  L1,L2
```

Als Attribute sind zugelassen:

PARAMETER für Konstante
DIMENSION zur Vereinbarung von Feldern
SAVE zum Sichern
POINTER für Zeiger
TARGET für Zeigerziele
PUBLIC, PRIVATE für Zugriffsrechte (Moduln)
ALLOCATABLE zur Vereinbarung dyn. Felder
EXTERNAL, INTRINSIC für Prozeduren als Parameter
INTENT, OPTIONAL für Prozedurparameter

Beispiele:

```
INTEGER, DIMENSION(1:10) :: vektor, x, y(9,7)
REAL          m1(10,10), m2(-5:+5)
REAL, ALLOCATABLE, DIMENSION(:,:) :: u, v, z(:)
CHARACTER (LEN=28), SAVE :: namens_liste
```

Datenstrukturen (Derived Types)

– Definition in TYPE-Blöcken

Beispiel:

```
TYPE PERSON
  CHARACTER(LEN = 45) :: NAME
  CHARACTER(LEN = 25) :: VORNAME
  INTEGER :: GEBURTSJAHR
END TYPE PERSON
```

– Komponenten: Felder mit explizit def. Gestalt und Zeiger, jedoch keine dynamisch erzeugbaren Felder

– Datenstrukturen innerhalb von Datenstrukturen sind erlaubt

– Deklaration mit Typanweisung

Skalar:

```
TYPE(PERSON) :: PERSONA
```

Feld:

```
TYPE(PERSON),DIMENSION(1:50) :: ABTEILUNG
```

Strukturkomponenten

Zugriff mit Prozentzeichen (%)
Beispiel:
PERSONA%NAME
ABTEILUNG(J)%VORNAME

Konstruktion von Strukturwerten und -konstanten

Eine Folge von Werten für die Komponenten ergibt einen Strukturwert

Wenn alle Komponenten konst. Ausdrücke sind, liegt eine Strukturkonstante vor

Beispiel:
PERSON(´SCHMITZ´,´FRANZ´,1937)

Operationen für Datenstrukturen

Vordefinierte Operation: Zuweisung (=) bei Typ-Übereinstimmung

Weitere Operationen durch eigene Operatoren

Zeiger in Fortran

- typgebunden

- kein direkter Zugriff und keine Rechenoperationen mit Zeigern erlaubt

- deklariert mit den Spezifikationen Typ und Rang, nicht nur Adresse

- Attribut: POINTER

- nur ohne Indexgrenzen deklarierbar

- sind undefiniert und können erst benutzt werden, wenn der Zeiger mit dem Speicher verknüpft wurde

- Zeiger werden auf 2 Arten mit dem Speicher verknüpft:

  1) durch Ausführung einer ALLOCATE-Anweisung
  2) durch Ausführung einer Zeigerzuweisung
     ZEIGER => OBJEKT

Beispiele

```
REAL, DIMENSION(:,:), POINTER :: in, out
...
ALLOCATE(in(-3:m, 0:9), STAT = fehler)

REAL, DIMENSION(1000,1000), TARGET :: a, b
...
out => a ; x = out(2,3)
```

Typ, Typparameter und Rang müssen übereinstimmen

Auf Objekte, die kein Zeiger sind, können Zeiger nur dann verweisen, wenn sie das Attribut TARGET haben

Anstelle von a kann in der Zeigerzuweisung ein Zeigerausdruck stehen

Felder

Operationen

– mit kompletten Feldern und
– mit Teilfeldern möglich

– arithmetische, boolesche und Zeichen-operationen auch auf Feldoperanden

– Gestalt der Operanden muß übereinstimmen (Anzahl der Dimensionen und ihre Ausdehnung)

Beispiel:

```
REAL, DIMENSION(10, 10) :: a, b, c
REAL :: x(5,6), y(5:9, 0:5), z(-4:0, 2:7)
REAL, DIMENSION(:,:), POINTER :: p
ALLOCATE (p(4:8, 3:8))
...
c = a + b
z = x + y * p
a = a + 7.5
```

Speicherklassen von Feldern

- statisch
  nur konstante Indexgrenzen

- dynamische Speicherzuweisung
  ALLOCATABLE oder POINTER

  ------- in Prozeduren ----------

- automatisch erzeugte Felder
  (lokale Variable einer Prozedur)
  Lebensdauer = Prozeduraufruf
  Felder mit explizit definierter Gestalt und variablen Indexgrenzen, die von aktuellen Parametern abhängen

- Feld mit übernommener Gestalt
  formale Prozedurparameter
  Indexgrenzen innerhalb der Prozedur sind nicht veränderbar

- feldwertige Funktionsergebnisse

Dynamisch erzeugbare Felder

Beispiel:

```
REAL, DIMENSION( : , : ), ALLOCATABLE :: a,b
ALLOCATE(a(-2 : m, 1 : 10),STAT = status)
...
DEALLOCATE(a, STAT = status)
```

Automatisch erzeugte Felder

- Felder, die weder Formalparameter noch Funktionsergebnis sind

- Vereinbarung mit explizit definierter Gestalt enthält variable Indexgrenzen, die von Parametern abhängen (z.B. aktuelle Prozedurparameter, Variable in COMMON und Modul)

- Felder werden automatisch bei Eintritt in die Prozedur angelegt (Hilfsfelder)

Beispiel:

```
SUBROUTINE RECHNE (feld1, laenge)
 COMMON/Feld_Grenzen/m,n
 REAL :: feld1 (laenge) ! Formalparameter
 REAL :: feld2 (3,laenge) ! autom. erzeugtes Feld
 REAL :: feld3 (m:n) ! autom. erzeugtes Feld
```

Feld mit übernommener Gestalt

- Formalparameter einer Prozedur

- Nur die Gestalt des Aktualparameters wird übernommen, nicht die Indexgrenzen

- Untere Indexgrenzen dürfen angegeben werden, sonst gleich 1

- Schnittstelle muß bekannt sein

Formalparameter: `REAL :: feld_f(0:,2:)`
oder `REAL :: feld_f(:,:)`

Aktualparameter: `REAL :: feld_a(0:9, 1:8)`

Felderzeuger

ist eine Folge von skalaren Werten
zwischen (/ und /)

Beispiel:

```
REAL, DIMENSION(4) :: X
X = (/ 3.2,4.01,6.5,2.3 /)
```

Ergebnis: X(1) = 3.2 X(2) = 4.01
X(3) = 6.5 X(4) = 2.3

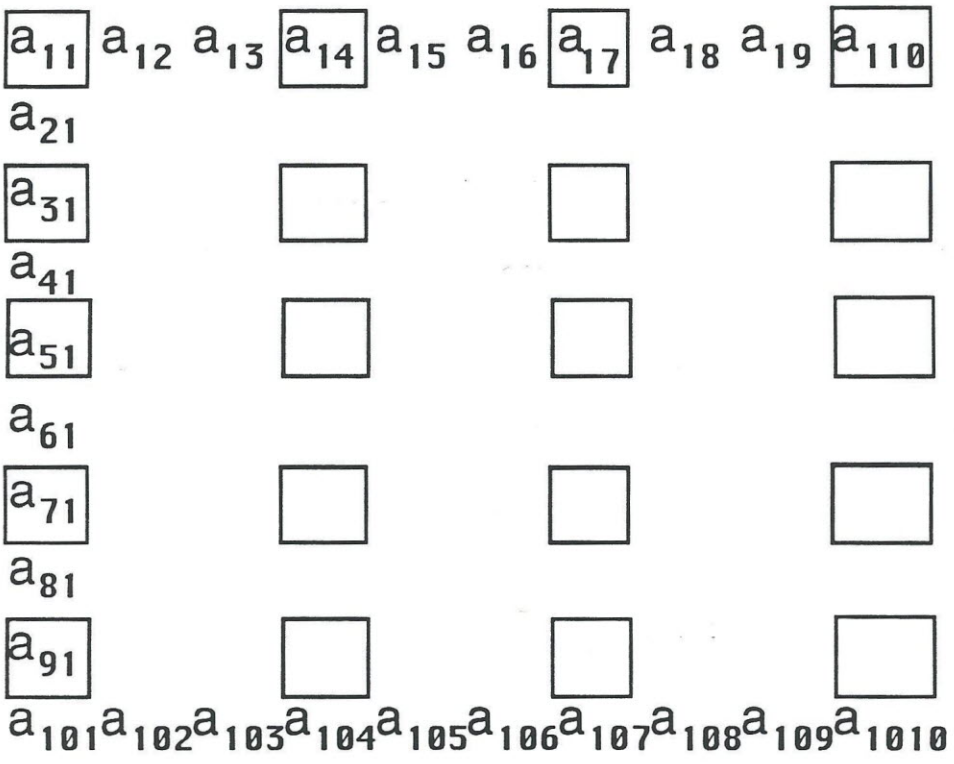
Beispiel mit impl. DO:

```
INTEGER, DIMENSION(1 : 51) :: IA
N = 100
IA = (/ 0, (I, I = 1, N, 2) /)
```

Teilfelder

Durch Triplexnotation wird ein rechteckiger Ausschnitt mit Lücken aus einem Feld ausgewählt

Beispiel:
REAL, DIMENSION(10, 10) :: a
Teilfeld a(1 : 10 : 2, 10 : 1 : -3)



Vektorindizes

Vektoren dürfen zur Indizierung verwendet werden:

$$A = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Mit U = (/ 3, 2 /) ist A(: , U) gleich $\begin{pmatrix} 7 & 4 \\ 8 & 5 \\ 9 & 6 \end{pmatrix}$

Teilfelder

```
REAL:: v(6:10, 1:3)
i=9; j=3

Teilfelder:

v( i, 1 : j )
v( 6 : i, j )
v( i , : )
v( 6 : i : 2 , j )
v( 10 : 6 : -1, j )
v( (/ 6 , 9 , 8 , 10 /), 2 )
v( 6, 1 : 3 : 2 )
v( 7 : 9 , : )
v( i : , 2 : 3 )
v( : i , 1 : j : 2 )
v( : : 2 , : )
```

Beachte: v(6, 1) ! Skalar
v(6:6, 1) ! Teilfeld (Rang 1)

Feldfunktionen

- Skalare, vordefinierte Elementarfunktionen können Feldargumente haben:
  SIN, COS, TAN, SQRT

- Neue Funktionen für Felder:
  DOTPRODUCT, MATMUL, MAXVAL

- Abfragefunktionen für Felder:
  ALLOCATED, LBOUND, UBOUND, SIZE, SHAPE

- Benutzer kann feldwertige Funktionen definieren:

```
FUNCTION feld_f (x, y)

REAL, DIMENSION(100, 100) :: feld_f,x,y

REAL feld_f(LBOUND(x,1):UBOUND(x,1),&
 LBOUND(x,2):UBOUND(x,2))

REAL,DIMENSION(:,:),POINTER :: feld_f
```

Maskierte Feldzuweisung

(Masked array assignment - WHERE)

Anweisung WHERE:

```
REAL,&
DIMENSION(1:9,1:20)::TEMP, DIFF_TEMP, DRUCK

WHERE(TEMP < 100.0) TEMP = TEMP - DIFF_TEMP
```

Anweisungsgruppe WHERE:

```
WHERE(TEMP > 50.0)
  DRUCK = DRUCK + 1.5
  TEMP  = TEMP - DIFF_TEMP
ELSEWHERE
  DRUCK = 1.0
END WHERE
```

- Nur Zuweisungen erlaubt

- Maske und Felder müssen in der Gestalt übereinstimmen

Modul

- neue Programmeinheit

- enthält Definitionen und Vereinbarungen für globale Datenobjekte, Typdeklarationen, Prozedur-Interface-Blöcke und Prozedurdefinitionen

- nimmt Vereinbarungen auf, damit alle anderen Programme darauf zugreifen können

- auch im Modul enthaltene Programme dürfen auf seine Vereinbarungen zugreifen

- ersetzt Funktionalität von COMMON und BLOCK DATA, ohne sich auf die Speicherabbildung zu beziehen

- Zugriff mit USE

Globale Felder mit dynamischer Speicherverwaltung

```
MODULE ARBEITSFELDER
 INTEGER :: N
 REAL, POINTER, SAVE :: A(:), B(:,:), C(:,:,:)
END MODULE ARBEITSFELDER

PROGRAM HAUPTPROGRAMM
 CALL KONFIGURIERE_FELDER
 CALL RECHNE
END PROGRAM HAUPTPROGRAMM

SUBROUTINE KONFIGURIERE_FELDER
 USE ARBEITSFELDER
 READ (INPUT, * ) N
 ALLOCATE(A(N), B(N,N), C(N, N, 2*N) )
END SUBROUTINE KONFIGURIERE_FELDER

SUBROUTINE RECHNE
 USE ARBEITSFELDER, ONLY : A, B, C
 ... ! Bearbeitung der Felder A, B und C
END SUBROUTINE RECHNE
```

Prozeduren

- Prozedurklassen:

  Externe Prozeduren
  Modulprozeduren
  Interne Prozeduren
  Vordefinierte Prozeduren
  (Intrinsic Procedures)

- Prozedurarten(Aufruf):

  Funktionen
  Subroutinen

- Prozedurschnittstelle(Interface):

  Schnittstellendefinition für den
  Prozeduraufruf

Interne Prozedur

Beispiel:

```
SUBROUTINE EXTERN
  REAL :: A, B, C
  ...
  CALL LOESCHEN
  ...
  CONTAINS
  SUBROUTINE LOESCHEN
  REAL X
  A = 0.0
  B = 0.0
  C = 0.0
  ...
  END SUBROUTINE LOESCHEN
END SUBROUTINE EXTERN
```

Die interne Prozedur LOESCHEN greift auf die Variablen des umgebenden Programmes zu

Sie kann am Ende eines Unterprogramms und eines Hauptprogramms stehen

Prozedur-Interface-Block

- beschreibt die Prozedurparameter

- dient der Kontrolle des Prozeduraufrufs

- enthält die Charakteristika:

  - Namen mit Attributen für jeden formalen Parameter
  - Operator bei Funktionen
  - Zuweisung bei Subroutinen

Beispiel:

```
INTERFACE
  SUBROUTINE EXT1(X, Y, Z)
  REAL,DIMENSION(100, 100) :: X, Y, Z
 END SUBROUTINE EXT1
END INTERFACE
```

macht aus einer unbekannten Schnittstelle der externen Subroutine EXT1 eine bekannte (überprüfbare) Schnittstelle

Neue Steueranweisungen

CASE-Anweisungsgruppe für INTEGER, CHARACTER, LOGICAL

Beispiel:

```
INTEGER FUNCTION SIGNUM (N)
 SELECT CASE (N)
  CASE( : -1)
   SIGNUM = -1
  CASE( 0 )
   SIGNUM = 0
  CASE( 1: )
   SIGNUM = 1
 END SELECT
END FUNCTION SIGNUM
```

CASE-Index: Liste von Werten und Intervallen, z.B. ("A" : "H", "Z")

a : b bedeutet: a <= CASE-Index <= b

Für fehlende CASE-Indizes kann CASE DEFAULT angegeben werden

DO-Anweisungsgruppe

- Laufanweisung: SCHLEIFE : DO 100, I = 1,10,1
  oder SCHLEIFE : DO 200, WHILE(SUM < MAX)

Beispiel:
```
N = 0
SCHLEIFE : DO 100, I = 1, 10, 1
 N = N + 1
100 END DO SCHLEIFE
```

- Schleifenname, Marke und Schleifensteuerung können entfallen

- EXIT beendet eine DO-Schleife

- CYCLE vermindert den Iterationszähler um 1 und erhöht die Laufvariable um die Schrittweite

NAMELIST

- Vereinbarung:
  NAMELIST /LISTE1/ A, B, C, /LISTE2/ E, F

- READ-Anweisung:
  READ(UNIT = 6, NML = LISTE1)

- Daten:
  &LISTE1 C = 17.32, B = 56.47/

Überholte Sprachmittel

- gekennzeichnet im Standard

- voraussichtlich nicht in die nächste Norm (Fortran 2000) übernommen

- Kennzeichnung durch Übersetzer

- dazu zählen:

Arithmetisches IF
REAL- und DOUBLE-PRECISION-Variable als Schleifenvariable
Mehrere DO-Schleifen, die von einer gemeinsamen Anweisung beendet werden
Das Ende einer DO-Schleife ist nicht END DO oder CONTINUE
Sprung auf eine END IF-Anweisung von außerhalb
RETURN i
PAUSE-Anweisung
ASSIGN und "assigned" GO TO
Zuweisung einer Marke einer FORMAT-Anweisung an eine ganzzahlige Variable mit der ASSIGN-Anweisung

Zusammenfassung

Unter den Erweiterungen gegenüber FORTRAN 77 sind besonders die folgenden hervorzuheben:

- Feldoperationen
  - Mit vollständigen Feldern und Teilfeldern
  - Vektoren als Indizes
- Dynamische Felder
- Zeiger
- Benutzerdefinierte Datentypen und Operatoren
- Modulkonzept für Daten und Prozeduren
- Verbesserte Portierbarkeit numer. Programme
- Prozeduren:
  - Modulprozeduren
  - Rekursive Prozeduren
  - Interne Prozeduren
  - Definition expliziter und generischer Schnittstellen für Prozeduren
- Parametrisierung vordefinierter Datentypen

- Verbesserte Ein-/Ausgabe
  - Ein-/Ausgabe mit Gruppennamen (NAMELIST)
  - Zeichenweise Ein- und Ausgabe
  - Ein- und Ausgabe ganzer Zahlen in binärer, oktaler und sedezimaler Darstellung
- Zusätzliche Zeichensätze sind erlaubt
- Bitmanipulationen mit vordef. Prozeduren
- Lexikalische Erweiterungen
  - Formatfreie Quellform
  - Kommentare in Anweisungszeilen
  - Mehrere Anweisungen in einer Zeile
  - Namen können bis zu 31 Zeichen lang sein
  - Operationszeichen für Relationen
  - INCLUDE-Zeile

- Neue Steueranweisungen für strukturierte Programmierung

- Viele neue vordefinierte Funktionen für Felder, Bits, numerische Auswertungen, für die Abfrage von Datum und Uhrzeit

- Fortran 90 enthält ein Konzept für die Evolution der Programmiersprache:
  - Überholte Sprachmittel sollen später wegfallen

Literatur

- Norm DIN EN 21 539 Fortran
  enthält ISO/IEC 1539 : 1991 (englische Fassung)
  Beuth Verlag, Burggrafenstr. 4-10
  1000 Berlin 30

- Metcalf, M.; Reid,J.: Fortran 90 Explained
  Oxford Univ. Press, Oxford

- W. Brainerd, C. Goldberg, J. Adams -
  Programmers Guide to Fortran 90
  Intertext Publications
  Mc Graw-Hill Book Company N.Y.
  ISBN 0-07-000248-7

- M. Heisterkamp, K.-H. Rotthäuser:
  Fortran 90 - Eine informelle Einführung
  B.I. Wissenschaftsverlag
  Mannheim / Wien / Zürich
  Nov. 1991, ISBN 3-411-15321-0

- W. Gehrke:
  Fortran 90 Referenz-Handbuch,
  Der neue Fortran-Standard
  Carl Hanser Verlag München Wien
  ISBN 3-446-16321-2