Vereinbarungen in F77 (bzw. F90)

Schreibweise der Vereinbarungsanweisungen

```
typ [(KIND=...)[, attribute] ::]  variable, ...
```

Skalare Datentypen / Konstanten

```
INTEGER ...
REAL ...
DOUBLE PRECISION ...
COMPLEX ...
LOGICAL ...
```

Zeichenketten

```
CHARACTER*n ...                    'Text', "Hello", 'Er hat''s', "Ich hab's"
```

Vereinbarungen in F77

Schreibweise der Vereinbarungsanweisungen

```
typ  variable, ...
```

Skalare Datentypen / Konstanten

```
INTEGER ...
REAL ...
DOUBLE PRECISION ...
COMPLEX ...
LOGICAL ...
```

Zeichenketten

```
CHARACTER*n ...          'Text', "Hello", 'Er hat''s', "Ich hab's"
```

Vereinbarungen in F77

Schreibweise der Vereinbarungsanweisungen

```
typ  variable, ...
```

Skalare Datentypen / Konstanten

`INTEGER ...`	0 , 1 , 2 , -1 , -2 , ...
`INTEGER*4 ...`	-2 147 483 648 ... 2 147 483 647
`INTEGER*2 ...`	-32 768 ... 32 767
`INTEGER*1 ... oder BYTE ...`	-128 ... 127
`REAL ...`	
`DOUBLE PRECISION ...`	
`COMPLEX ...`	
`LOGICAL ...`	

Zeichenketten

`CHARACTER*n ...`	`'Text', "Hello", 'Er hat's', "Ich hab's"`

Vereinbarungen in F77

Schreibweise der Vereinbarungsanweisungen

```
typ  variable, ...
```

Skalare Datentypen / Konstanten

```
INTEGER ...
REAL ...                                  -1. , 0.1, .2 , 1e-4 , -1.0e3 , ...
  REAL*4 ...
  REAL*8 ... oder DOUBLE PRECISION
  REAL*16 ... (als "Zugabe")
DOUBLE PRECISION ...                      -1.d0 , 2d-1 , 1d-4 , -1.0d123 , ...
COMPLEX ...
LOGICAL ...
```

Zeichenketten

```
CHARACTER*n ...                  'Text', "Hello", 'Er hat''s', "Ich hab's"
```

Vereinbarungen in F77

Schreibweise der Vereinbarungsanweisungen

```
typ  variable, ...
```

Skalare Datentypen / Konstanten

```
INTEGER ...
REAL ...
DOUBLE PRECISION ...
COMPLEX ...                          (-1.,0.) , (0.0,1.0) , (1e-4,0.) , ...
  COMPLEX*8 ...
  COMPLEX*16 ...                                    (0d0,1d0) , ...
  COMPLEX*32 ...
LOGICAL ...
```

Zeichenketten

```
CHARACTER*n ...          'Text', "Hello", 'Er hat''s', "Ich hab's"
```

Vereinbarungen in F77

Schreibweise der Vereinbarungsanweisungen

```
typ  variable, ...
```

Skalare Datentypen / Konstanten

```
INTEGER ...
REAL ...
DOUBLE PRECISION ...
COMPLEX ...
LOGICAL ...                                    .TRUE. , .FALSE.
```

Zeichenketten

```
CHARACTER*n ...          'Text', "Hello", 'Er hat''s', "Ich hab's"
```

Vereinbarungen in F77

Schreibweise der Vereinbarungsanweisungen

```
typ  variable, ...
```

Skalare Datentypen / Konstanten

```
INTEGER ...
REAL ...
DOUBLE PRECISION ...
COMPLEX ...
LOGICAL ...
```

Zeichenketten

```
CHARACTER*n ...          'Text', "Hello", 'Er hat''s', "Ich hab's"
CHARACTER str1*n, str2*m
```

Vereinbarungen in F77

Benannte Konstanten (PARAMETER-Anweisung) (Bsp.)

```
REAL*8 NULL, PI
PARAMETER (NULL=0.d0, PI=3.14159265358979323385d0)
CHARACTER*(*) Text
PARAMETER (Text="Hello World!")
INTEGER Max_Length
PARAMETER (Max_Length=100000)
```

Anfangswertzuweisung (DATA-Anweisung) (Bsp.)

```
DOUBLE PRECISION pi, e
DATA pi /3.14159265358979323385d0/
DATA e /2.71828182845904523354d0/
```

Vereinbarungen in F77

Anfangswertzuweisung (DATA-Anweisung) (Bsp.)

```
DOUBLE PRECISION pi, e
DATA pi /3.14159265358979323 85d0/
DATA e /2.71828182845904523 54d0/
```

(unschöne) alternative Mischform:

```
DOUBLE PRECISION pi /3.14159265358979323 85d0/
DOUBLE PRECISION e /2.71828182845904523 54d0/
```

neue Variante in F90:

```
DOUBLE PRECISION ::  pi = 3.14159265358979323 85d0
DOUBLE PRECISION ::  e = 2.71828182845904523 54d0
```

Vereinbarungen in F77

Alternative Schreibweise von Konstanten

Numerische Konstanten können in anderen Zahlensystemen angegeben werden (für ganzzahlige Konstanten manchmal sinnvoll, sonst kaum):

Beispiele:			
	binär	`B'110'`	$(= 6)$
	oktal	`O'107'`	$(= 71)$
	hexadezimal	`Z'AFFE'`	$(= 45054)$

Für Zeichenkettenkonstanten gibt es (noch) die veraltete Form als Hollerith-Konstante ohne Anführungszeichen (Verwendung in der `FORMAT`-Anweisung zur Ausgabe von Text, aber auch in der `DATA`-Anweisung toleriert)

Beispiel:

```
CHARACTER*12 str
DATA str / 12HHello World!  /
```

Vereinbarungen in F77

Implizite Typvereinbarung

Wenn nicht explizit vereinbart, erhält jede Variable innerhalb einer Programmeinheit einen Standardtyp zugeordnet, der sich nach dem Anfangsbuchstaben des Variablennamens richtet:

`I,J,K,L,M,N`	$\Rightarrow$	`INTEGER`
`A-H,O-Z`	$\Rightarrow$	`REAL`

Das gilt auch für Funktionsnamen, die nicht explizit mit einem bestimmten Typ vereinbart wurden !

Vereinbarungen in F77

IMPLICIT-Anweisung

Eine abweichende Zuordnung zwischen Anfangsbuchstabe und Typ kann man auch selbst festlegen:

```
IMPLICIT typ (buchstabenliste)
```

Bsp.:

```
IMPLICIT DOUBLEPRECISION (D-H,O-Y)
IMPLICIT LOGICAL (L)
IMPLICIT COMPLEX (A-C,Z)
```

Empfehlung: `IMPLICIT NONE`

- Diese Anweisung verhindert jegliche implizite Typvereinbarung.
- Jede Variable muss explizit vereinbart sein.
- Dringend zu empfehlen, denn Tippfehler bei Variablennamen werden damit meist sofort erkannt.

Vereinbarungen in F77

Vereinbarung von Feldern (Arrays) in F77

Verschiedene Schreibweisen:

```
typ vname(dim1,dim2,...,dimn), ...
```

oder

```
typ vname, ...
DIMENSION vname(dim1,dim2,...,dimn)
```

Dimensionsangaben (dim)

- Für jede Dimensionsangabe entweder Anzahl der Elemente,
  z.B. `REAL vektor(200), Matrix(200,200)`
  dann werden die Elemente jeweils mit 1 beginnend indiziert
- oder als Indexbereich `(von:bis)`
  z.B. `INTEGER Z(0:9), A(200,-2:2)` oder

```
CHARACTER*2 wochentag(0:6)
DATA wochentag /'So','Mo','Di','Mi','Do','Fr','Sa'/
```

Vereinbarungen in F90

Grundtypen

`INTEGER, REAL, COMPLEX, LOGICAL, CHARACTER`

sind vordefinierte Grundtypen (*intrinsic data types*);
`DOUBLEPRECISION` ist nur eine Variante des Grundtyps `REAL`:

```
REAL (KIND=8) ::  X, Y, Z
```

oder

```
REAL (8) ::  X, Y, Z
```

Nach dem `KIND`-Selektor kann eine Attributliste folgen, z. B.

```
PARAMETER, DIMENSION(dim1,...), ALLOCATABLE,
POINTER, TARGET, INTENT(...), EXTERNAL, ...
```

Vereinbarungen in F90

Schreibweise der Vereinbarungen wie in F77 kann auch in F90 verwendet werden, aber zusätzlich diverse andere ...

abweichende F90-Schreibweise — Beispiele

```
INTEGER (KIND=2), PARAMETER ::  ASCII=1_2, LNG=1000_2
REAL (KIND=16),DIMENSION(LNG,0:1) ::  XL, YL
INTEGER, EXTERNAL ::  fakultaet
CHARACTER (LEN=12,KIND=ASCII) ::  Txt=ASCII_"Hello World!"
```

Passende KIND-Werte automatisch bestimmen

```
INTEGER,PARAMETER   ::  kurz=selected_int_kind(2)
INTEGER (KIND=kurz) ::  kvar=99_kurz
!  INTEGER-Typ fuer mindestens 2-stellige (Dezimal-)Zahlen
REAL(KIND=selected_real_kind(4,3)) ::  a,b
!  Wertebereich fuer a,b mindestens -1000 ...1000 und
!  Genauigkeit mindestens 4 Dezimalziffern
```

Vereinbarungen in F90

`selected_int_kind(r)`

Minimalforderung an den Integer-Typ: Zahlbereich umfasst $-10^r \ldots 10^r$. Mit

```
write(*,*) KIND(m), RANGE(m)
```

kann man den Wert des KIND-Selektors und den Zahlbereich für eine Integer-Variable `m` anzeigen.

Beispiel: `m` sei vereinbart mit `selected_int_kind(r)`

`r = 1` $\rightarrow$ `KIND=1, RANGE=2,`
`r = 3` $\rightarrow$ `KIND=2, RANGE=4`
`r = 5` $\rightarrow$ `KIND=4, RANGE=9,`
`r = 10` $\rightarrow$ `KIND=8, RANGE=18`
`r = 19` $\rightarrow$ `KIND=16, RANGE=38`

Vereinbarungen in F90

`selected_real_kind(p,r)`

Minimalforderung an den Real-Typ: Zahlbereich umfasst $\pm 10^{-r} \ldots \pm 10^{r}$, mit einer Genauigkeit von p Dezimalstellen. Mit

```
write(*,*) KIND(x), RANGE(x), PRECISION(x)
```

kann man den Wert des REAL-Selektors, den Zahlbereich und die Genauigkeit für eine Real-Variable `x` anzeigen.

Beispiel: `x` sei vereinbart mit `selected_real_kind(p,r)`

```
(p,r) = (6,30)  → KIND=4,  RANGE=37,   PRECISION=6,
(p,r) = (7,30)  → KIND=8,  RANGE=307,  PRECISION=15
(p,r) = (17,40) → KIND=10, RANGE=4931, PRECISION=18,
(p,r) = (20,40) → KIND=16, RANGE=4931, PRECISION=33
```

Vereinbarungen in F90

Arrays in F90 vereinbaren

- Dimension wie in F77 als Zusatz zum Variablennamen

```
INTEGER ([KIND=]2) ::  L(10)
CHARACTER ([LEN=]20) ::  d(7)
```

- als Attribut in der Typvereinbarung

```
INTEGER,DIMENSION(3,3) ::  e3
REAL,ALLOCATABLE,DIMENSION(:)  ::  a,b
```

- einzelne Deklarationen (lieber vermeiden!)

```
REAL ::  feld
ALLOCATABLE ::  feld(:,:)
```

Dynamische Speicherplatzzuweisung und -freigabe:

```
ALLOCATE(a(-100:100))
ALLOCATE(b(n+2))
ALLOCATE(feld(0:99,n))
    ..................
DEALLOCATE(a,b,feld)
```

Im Vereinbarungsteil sind nur Platzhalter für Dimensionsangaben vorhanden, dann `ALLOCATE / DEALLOCATE` im Anweisungsteil als ausführbare Anweisungen

Vereinbarungen in F90

Pointer (eigentlich nur Alias)

- Es handelt sich **nicht** um Variable, deren Wert eine Speicheradresse ist.
- `POINTER` ist ein Attribut, das eine Variable (nach Zuweisung) zu einem Alias für eine andere Variable macht, die das Attribut `TARGET` besitzt.
- Beispiel 1:

```
INTEGER,POINTER ::  a, b(:,:)
INTEGER,TARGET ::  c, d
...
a => c    ! jetzt ist a ein Alias für c
...
a => d    ! und nun für d
```

Vereinbarungen in F90

Pointer (eigentlich nur Alias)

- Es handelt sich **nicht** um Variable, deren Wert eine Speicheradresse ist.
- `POINTER` ist ein Attribut, das eine Variable (nach Zuweisung) zu einem Alias für eine andere Variable macht, die das Attribut `TARGET` besitzt.
- Beispiel 2:

```
REAL,POINTER,DIMENSION(:,:)  ::  oben, unten, links
REAL,TARGET,DIMENSION(10,10) ::  matrix
...
oben  => matrix(1:5,1:5)    ! Diagonalblock links oben
...
unten => matrix(6:10,6:10)  ! Diagonalblock rechts unten
...
links => matrix(1:10,1:5)   ! erste 5 Spalten
```

Vorsicht! – Es ist nicht definiert, wie der Compiler mit solchen Teilfeldern umzugehen hat. Je nach Verarbeitung kann es notwendig sein, dass intern solche Teile temporär im Arbeitsspeicher hin und her kopiert werden.
Das kostet Zeit und Speicherplatz!

Vereinbarungen in F77/F90

EQUIVALENCE-Anweisung

```
EQUIVALENCE (a,b,c), (x,y)
```

- Vergleichbar zum Pointer-Attribut, aber statisch definiert
- Verschiedene Variablennamen (und -typen) – gleicher Speicherplatz
- nur für interne Variable einer Programmeinheit,
  nicht für formale Argumente in der Rufzeile von UP

Beispiele (mehr oder weniger sinnvoll)

```
COMPLEX c
REAL x(2), cR, cI
EQUIVALENCE (c,x(1),cR), (x(2),cI)
```

C →
x →

cR	cI
x(1)	x(2)

```
REAL z
INTEGER n
EQUIVALENCE (n,z)
```

dann als Test:

```
READ(*,*) z
WRITE(*,*) n
```

(oder umgekehrt)

Vereinbarungen in F77/F90

COMMON-Blöcke

```
COMMON /bezeichnung/ variablenliste,...
```

- Anweisung steht zweckmäßigerweise am Ende des Vereinbarungsteils
- Variablen werden zu einem *gemeinsamen Speicherbereich* zusammengefasst, der so in verschiedenen Programmeinheiten zugreifbar wird (überall, wo derselbe Common-Block vereinbart ist)
- Speicherung erfolgt in angegebener Reihenfolge der Variablen
- `bezeichnung` wird als globaler Name behandelt, ähnlich wie Namen von Unterprogrammen oder Funktionen
- Die Namen der aufgelisteten Variablen müssen in verschiedenen Programmeinheiten nicht übereinstimmen (wäre aber schlechter Programmierstil)
- verschiedene Bezeichnungen $\Rightarrow$ mehrere Common-Blöcke.
- Basis-Fortran $\Rightarrow$ nur **ein** (unbenannter!) COMMON-Block

Vereinbarungen in F77/F90

Was sonst noch vorkommen kann

- `INCLUDE 'dateiname'` – ein Stück Quelltext vor dem Übersetzen einbinden, z.B. Vereinbarung von Variablen, die zu einem Common-Block zusammengefasst werden, damit diese in verschiedene Unterprogramme eingebunden werden können
  (und bei evtl. Änderungen keine der betroffenen Quellen *vergessen* wird).
- `EXTERNAL name1` oder `INTRINSIC name2`, bzw. in F90 auch `typ, EXTERNAL :: name1` oder `typ, INTRINSIC :: name2`
  `name1` = Name eines FUNCTION-Unterprogramms, der als aktueller Parameter an ein Unterprogramm übergeben werden soll, in dem diese Funktion dann gerufen wird; bzw.
  `name2` = Name einer Fortran-Standardfunktion, die anstelle eines eigenen Funktionsunterprogramms als aktueller Parameter übergeben wird.
- ...