Anweisungen in Fortran - Wertzuweisung

Wertzuweisung, Ausdrücke

`variable = ausdruck`

- arithmetischer, logischer oder Zeichenketten-Ausdruck,
- Typ des Ergebnisses sollte auf den Typ der linksstehenden Variablen konvertierbar sein.

Arithmetische Ausdrücke

Operatoren mit absteigender Priorität:

`( )`	Klammerung (höchste Priorität)
`**`	Potenzierung (`x**y` $\rightarrow x^y$)
`*,/`	Punktrechnung geht vor
`+,-`	Strichrechnung

Bei gleicher Priorität aufeinanderfolgender Operatoren wird der Ausdruck von links nach rechts abgearbeitet

```
a/b * c-d * e-f = (((a/b)*c) - (d*e)) - f
```

Ausnahme: a^{b^c} = `a**b**c = a**(b**c)`

Anweisungen in Fortran - Wertzuweisung

Auswertung arithmetischer Ausdrücke - Typkonvertierung

- Mit `*,/,+,-` werden jeweils 2 Operanden gleichen Typs miteinander verknüpft, das (Teil-)Ergebnis hat den gleichen Typ wie die Operanden
- Bei verschiedenen Typen der beiden Operanden wird auf den allgemeineren Typ konvertiert in der Richtung:

  `INTEGER`→`REAL`→`COMPLEX` bzw. `typ*2`→`typ*4`→`typ*8`
- Der gesamte Ausdruck hat dann den allgemeinsten aller bei den Operanden vorkommenden Typen.
  Er wird erst zum Schluss in den Typ der links stehenden Variablen konvertiert.
- Dadurch kann die Reihenfolge von Operanden starke Auswirkungen auf das Ergebnis haben.
- Beachte: `real**integer` meist besser als `real**real`
  (ersteres: Multiplikation, letzteres: Logarithmen)

Anweisungen in Fortran - Wertzuweisung

Auswertung arithmetischer Ausdrücke - Beispiele

Es gelten die impliziten Typvereinbarungen, sowie
`IMPLICIT DOUBLEPRECISION (d)`

Anweisung	Wert des Ausdrucks	Wert nach Zuweisung
`x = 3/4`	0	x=0.0
`x = 3./4`	0.75	x=0.75
`j = 3./4`	0.75	j=0
`dx= 1./3`	0.333333343	dx=0.33333334326744080
`dx= 1/3d0`	0.3333333333333331	dx=0.3333333333333331
`x = 1`	1	x=1.0
`y = 3/4*x`	0*x= 0.0	y=0.0
`y = x*3/4`	0.75	y=0.75

Es kann sinnvoll sein, Variable explizit in einen anderen Typ zu konvertieren (vgl. Standardfunktionen), z.B. `DBLE(x)`.

Versteckte Typumwandlungen kann man sonst im Quelltext leicht übersehen.

Anweisungen in Fortran - Wertzuweisung

Zeichenkettenoperationen

- Character-Variable haben eine feste (vereinbarte) Länge
- Bei Zuweisung an eine Variable wird rechts mit Leerzeichen aufgefüllt oder rechts abgeschnitten

  Beispiel:
  ```
  character zk*3, str*20
  zk = 'x'              →  'x  '
  zk = 'abcdef'         →  'abc'
  ```

- Substrings `str(von:bis)` können als *Quelle* oder auch als *Ziel* einer ZK-Operation verwendet werden:

  Beispiel:
  ```
  str='Kapitel *:'      →  'Kapitel *:       '
  Lmax = Len(str)       →  20
  L = Len_Trim(str)     →  10
  str(L-1:L-1) = 'A'    →  'Kapitel A:       '
  zk = str(:3)          →  'Kap'
  zk = str(3:5)         →  'pit'
  zk = str(5:)          →  'tel' (rechts abgeschnitten)
  ```

Anweisungen in Fortran - Wertzuweisung

Zeichenkettenoperationen

- Verkettungsoperation //

```
CHARACTER*5 H, W
CHARACTER*80 zeile
DATA H /'Hello'/, W /'World'/
zeile = H // ' ' // W // '!'        →    'Hello World!'
```

- Konvertierungen zwischen Zeichen und ASCII-Code

```
CHARACTER*1 c, esc
k = ICHAR('1')        →   k = 49
c = CHAR(k+1)         →   c='2'
esc = CHAR(27)        →   ASCII-Steuerzeichen Escape
```

- Suche nach bestimmten Zeichen in einer Zeichenkette

```
leer = INDEX(zeile,' ')     →   leer = 6
komma = INDEX(zeile,',')    →   komma = 0
```

Anweisungen in Fortran - Wertzuweisung

Logische Ausdrücke

arithmetischer Vergleich

ab Fortran IV		ab F90
`A1 .LT. A2`	$<$	<
`A1 .GT. A2`	$>$	>
`A1 .LE. A2`	$\leq$	<=
`A1 .GE. A2`	$\geq$	>=
`A1 .EQ. A2`	$=$	==
`A1 .NE. A2`	$\neq$	/=

Verknüpfungen: `.NOT. logA`

```
logA1   .AND.   logA2
logA1    .OR.   logA2
logA1   .EQV.   logA2
logA1  .NEQV.   logA2
```

Zeichenkettenvergleich

lexikalische Funktionen

```
LLT(str1,str2)
LGT(str1,str2)
LLE(str1,str2)
LGE(str1,str2)
```

Ordnungsrelation nach ASCII-Code von links nach rechts, so z. B.

```
' ' < '+' < '1' < '9' < 'A'
'B' < 'Z' < '_' < 'a' < 'z'
```

Logische Ausdrücke überwiegend für `IF (...)` oder `WHILE (...)` verwendet, aber auch bei Zuweisung an `LOGICAL`-Variable.

Anweisungen in Fortran - Wertzuweisung

Array-Ausdrücke (ab F90 zugelassen)

- Operationen mit Feldern als Operanden werden elementweise ausgeführt.
  (Keine Matrixmultiplikation im Sinne der Linearen Algebra)
- Die Felder müssen (deshalb) mit gleicher Dimension vereinbart sein.
- Skalare Operanden werden auf jedes Feldelement angewendet.
- Mit der `WHERE`-Anweisung (bzw. dem `WHERE`-Konstrukt) kann man Operationen auf ausgewählte Elemente eines Feldes einschränken:

```
REAL,DIMENSION(10,10) :: a,b,c
...
WHERE (a>b) a = -1.0                    (WHERE-Anweisung)
WHERE (b>c)
  b = -1.0
ELSEWHERE                               (WHERE-Konstrukt)
  b = 1.0
END WHERE
```

Anweisungen in Fortran - Ablaufsteuerung

Ablaufsteuerung (global)

- `PAUSE [prompt]` (veraltet, *deleted feature*)
  Ausgabe von `PAUSE prompt` (`prompt`: Character-Konstante oder max. 5-stellige Zahl) und Warten auf Reaktion des Nutzers. Abbruch, wenn nicht `'go'` eingegeben wird.
- `STOP [msg]`
  Ausgabe von `STOP msg` (`msg`: Character-Konstante oder max. 5-stellige Zahl) und sofortiges Beenden des Programms (Notausgang!)
- `RETURN`
  Beendet das aktuelle (Funktions-)Unterprogramm, Rückkehr ins rufende Programm.
  Wenn im Quelltext nicht vorhanden, wird das `RETURN` vor der `END`-Anweisung automatisch angenommen.

Anweisungen in Fortran - Ablaufsteuerung

Sprunganweisung (obsolet – d. h. besser vermeiden)

- unbedingte Verzweigung,

```
      GOTO marke
      ...
marke CONTINUE
```

Fortsetzung mit der Anweisung, die durch `marke` markiert ist. (`marke` ist eine vorzeichenlose, max. 5-stellige Zahl)
Die leere Anweisung `CONTINUE` wird als *Markierungspunkt* im Programm benutzt.

Spezialvarianten

- berechnetes GOTO

```
i = ...          ∈ {1,...,n}
GOTO (m1,m2,...,mn), i
```

Sprungziel ist i-te Marke

- assigned GOTO

```
ASSIGN marke TO i
GOTO i [,(m1,m2,...,mn)]
```

Sprungziel zur Laufzeit festgelegt

Anweisungen in Fortran - Ablaufsteuerung

Verzweigung, IF-Anweisung(en)

- arithmetisches IF (Urversion, hardwarenah)

  `IF (arithm_ausdruck) m1,m2,m3`

  Je nach Wert a des Ausdrucks Sprung zu `m1` ($a < 0$), `m2` ($a = 0$), `m3` ($a > 0$).

- logisches IF (ab Fortran IV)

  `IF (log_ausdruck) anweisung`

  Bedingte Ausführung **einer** Anweisung
  (bei komplizierteren Aktionen meist `GOTO`- oder `CALL`-Anweisung)

- Block-IF (ab Fortran 77)

```
IF (log_ausdruck) THEN
 anweisung(en)
ELSE IF (log_ausdruck) THEN      (Mehrfach-Bedingungen)
 anweisung(en)
ENDIF
```

  Übliche `IF-THEN-ELSE`-Konstruktion (`ELSE`-Zweig kann fehlen)

Anweisungen in Fortran - Ablaufsteuerung

Verzweigung, SELECT-CASE-Anweisung

- Fallunterscheidung nach Auswertung eines skalaren Ausdrucks (`integer`,`logical`,`character`)

```
select case (skalarer_ausdruck)
case(werte1)
  anweisungen1
case(werte2)
  anweisungen2
...
case default
  anweisung...      falls keiner der Werte zutrifft
end select
```

- Als `werte` sind (disjunkte) Aufzählungen oder Intervalle möglich, z. B. `case(1,4,9)` oder `case(:-1)` (d. h. $\texttt{ausdruck} \leq -1$)
- Die Anweisungen nach der ersten passenden Werteliste werden ausgeführt.

Anweisungen in Fortran - Ablaufsteuerung

Schleifen, DO-Anweisung(en)

- Zählschleife (klassisch)

```
      DO 10 k = kvon, kbis [, kstep]
       anweisungen
10     CONTINUE
```

vor F77: kstep > 0 (oder andere ausführbare Anweisung)

- While-Schleife (F90 $\rightarrow$ F77)

```
DO WHILE (logischer_ausdruck)
 anweisungen
END DO
```

(mit Einfluss auf Wert des log. Ausdr.)

- „unendliche“ Schleife

```
DO                          ! infinite loop
 anweisungen
 IF (log_ausdr) EXIT
 anweisungen
END DO
```

Schleife verlassen

Anweisungen in Fortran - Ablaufsteuerung

Schleifen, DO-Anweisung(en)

- Zählschleife (F90 $\rightarrow$ F77)

```
DO k = kvon, kbis [, kstep]
 anweisungen
END DO
```

vor F77: kstep > 0
seit F77: kstep $\neq 0$

- While-Schleife (F90 $\rightarrow$ F77)

```
DO WHILE (logischer_ausdruck)
 anweisungen
END DO
```

(mit Einfluss auf Wert des log. Ausdr.)

- „unendliche“ Schleife

```
DO                          ! infinite loop
 anweisungen
 IF (log_ausdr) EXIT
 anweisungen
END DO
```

Schleife verlassen

Anweisungen in Fortran - Ablaufsteuerung

Schleifen, DO-Anweisung(en)

- Zählschleife (F90 → F77)

```
DO k = kvon, kbis [, kstep]
 anweisungen
END DO
```

vor F77: kstep > 0
seit F77: kstep ≠ 0

- While-Schleife (F90 → F77)

```
DO WHILE (logischer_ausdruck)
 anweisungen
END DO
```

(mit Einfluss auf Wert des log. Ausdr.)

- „unendliche“ Schleife

```
DO                          ! infinite loop
 anweisungen
 IF (log_ausdr) EXIT
 anweisungen
END DO
```

Schleife verlassen

Anweisungen in Fortran - Ablaufsteuerung

Schleifen, DO-Anweisung(en)

- Zählschleife (F90 → F77)

```
DO k = kvon, kbis [, kstep]
 anweisungen
END DO
```

vor F77: kstep > 0
seit F77: kstep $\neq 0$

- While-Schleife (F90 → F77)

```
DO marke WHILE (logischer_ausdruck)
 anweisungen
marke CONTINUE
```

(mit Einfluss auf Wert des log. Ausdr.)

- „unendliche“ Schleife

```
DO                          ! infinite loop
 anweisungen
 IF (log_ausdr) EXIT
 anweisungen
END DO
```

Schleife verlassen

Anweisungen in Fortran - Ablaufsteuerung

Schleifen, DO-Anweisung(en)

- Zählschleife (F90 → F77)

```
DO k = kvon, kbis [, kstep]
 anweisungen
END DO
```

vor F77: kstep > 0
seit F77: kstep $\neq 0$

- While-Schleife (F90 → F77)

```
DO WHILE (logischer_ausdruck)
 anweisungen
END DO
```

(mit Einfluss auf Wert des log. Ausdr.)

- „unendliche“ Schleife

```
DO                          ! infinite loop
 anweisungen
 IF (log_ausdr) EXIT
 anweisungen
END DO
```

Schleife verlassen

Anweisungen in Fortran - Ablaufsteuerung

Schleifen – Besonderheiten, Probleme

- vor F77: **nichtabweisende** Schleifen (mind. 1 Durchlauf)
- vor F77: nur positive Schrittweite (negative Schrittweite durch Rückwärtszählen einer anderen Variablen innerhalb der Schleife)
- *„Deleted feature“*: F77 erlaubte (vorübergehend) Zählschleifen mit `REAL`-Argumenten, z. B. `DO x = 0.1, 1.5, 0.2`

  Die vorher berechnete Anzahl der Durchläufe (hier: $\frac{1.5-0.1}{0.2}$) konnte von Rundungsfehlern abhängen.
- Laufvariable der Zählschleife darf im Inneren der Schleife nicht verändert werden (Anzahl der Durchläufe wird evtl. vorher berechnet).
- Der Wert der Laufvariablen ist **nach** vollem Durchlauf der Schleife **unbestimmt**. Beim Verlassen der Schleife mit `GOTO`- oder `EXIT`-Anweisung hat die Variable den aktuellen Zählerwert.
- Sprunganweisungen ins Innere einer Schleife sind nicht erlaubt.

Anweisungen in Fortran - Ablaufsteuerung

Schleifen – Besonderheiten, Probleme

- vor F77: **nichtabweisende** Schleifen (mind. 1 Durchlauf)
- vor F77: nur positive Schrittweite (negative Schrittweite durch Rückwärtszählen einer anderen Variablen innerhalb der Schleife)
- *„Deleted feature“*: F77 erlaubte (vorübergehend) Zählschleifen mit `REAL`-Argumenten, z. B. `DO x = 0.1, 1.5, 0.2`

  Die vorher berechnete Anzahl der Durchläufe (hier: $\frac{1.5-0.1}{0.2}$) konnte von Rundungsfehlern abhängen.
- Laufvariable der Zählschleife darf im Inneren der Schleife nicht verändert werden (Anzahl der Durchläufe wird evtl. vorher berechnet).
- Der Wert der Laufvariablen ist **nach** vollem Durchlauf der Schleife **unbestimmt**. Beim Verlassen der Schleife mit `GOTO`- oder `EXIT`-Anweisung hat die Variable den aktuellen Zählerwert.
- Sprunganweisungen ins Innere einer Schleife sind nicht erlaubt.

Anweisungen in Fortran - Ablaufsteuerung

Schleifen – Besonderheiten, Probleme

- Problematisch sind geschachtelte Schleifen mit **gleicher** Endemarke:

```
      do 10 i=1,n
        ...
        goto 10                    Sprungziel nicht im selben Block
        ...
        do 10 j=1,m
          ...
          goto 10                  Sprungziel im selben Block
          ...
10    continue
```

- Besser mit 2 verschiedenen Marken klare Struktur schaffen.

Anweisungen in Fortran - Ablaufsteuerung

Schleifen – Besonderheiten, Probleme

- Problematisch sind geschachtelte Schleifen mit **gleicher** Endemarke:

```
      do 20 i=1,n
        ...
        goto 20
        ...
        do 10 j=1,m
          ...
          goto 10
          ...
10      continue
20    continue
```

- Besser mit 2 verschiedenen Marken klare Struktur schaffen.

Anweisungen in Fortran - Ablaufsteuerung

Schleifen – Spezielle Steueranweisungen

- `CYCLE` – Sprung ans Ende der aktuellen (inneren) Schleife, nächster Durchlauf
- `EXIT` – vorzeitiges Verlassen der aktuellen Schleife
- Bei geschachtelten Schleifen kann man mit speziellen *Construct-Namen* angeben, auf welche Schleife sich `CYCLE` oder `EXIT` beziehen sollen.

```
outer:  do i=1,n
  ...
  inner:  do j=i,n
    ...
    if (...)  EXIT outer          → verlasse gesamte Schleife
    if (...)  EXIT                → verlasse nur innere Schleife
    if (...)  CYCLE     → überspringe Rest der inneren Schleife
    ...
  end do inner
end do outer
```

Anweisungen in Fortran - Ablaufsteuerung

Schleifen – Implizites DO

- Spezielle Aufzählung von Elementen mittels Laufindex

  ```
  (ausdruck, k=kvon,kbis[,kstep])
  ```

  (Anwendung bei `READ` und `WRITE`)

- Beispiel:

  ```
  REAL A(1000,1000)
  ...
  DO i=1,n                                  zeilenweise
    WRITE(*,*) (A(i,j),j=1,m)               je m Spalten
  END DO
  ```

  auch geschachtelt:

  ```
  WRITE(*,*) ((A(i,j),j=1,m),i=1,n)
  ```

- Beachte: Die Anweisung `WRITE(*,*) A` entspricht der Ausgabe:

  ```
  WRITE(*,*) ((A(i,j),i=1,1000),j=1,1000)   spaltenweise
  ```