Programmierung Paralleler Systeme

3. Programmiermodell “Single Programm Multiple Data”

3.1 Fortran95

3.1.7 Feldverarbeitung

3.1.8 Dynamische Speicherverwaltung

3.1.9 Bedingte Feldzuweisung

Claus-Uwe Linster

IMMD IV, Lehrstuhl für Betriebssysteme
Universität Erlangen-Nürnberg

linster@informatik.uni-erlangen.de

3.1.7 Feldverarbeitung

- Allgemeine Form der Feldspezifikation:

  `Syntax:(dim [,dim]...)      max. 7 Dimensionen (Rang)`

  - mit `dim = ug : og` wird jeweils die Untergrenze (ug) und die Obergrenze (og) festgelegt, wobei jeweils nach der Art des Feldes, die Grenzen angegeben werden können oder müssen bzw. fehlen dürfen oder müssen.
  - Terminologie
    - die **Art eines Feldes** wird im wesentlichen durch seine Attribute und Dimensionsspezifikation festgelegt. Man unterscheidet:
      - Felder mit expliziter Gestalt
      - Felder mit übernommener Gestalt
      - Felder mit übernommener Größe
      - automatische Felder und
      - dynamische Felder

Feldverarbeitung

- Terminologie (cont)
  - die Anzahl der Indexgruppen bestimmt die **Dimension** oder den **Rang** eines Feldes, wobei das “eigentliche” Feld den Rang 2 hat; also 2-dimensional ist.
    Es hat sich aber eingebürgert auch die höherdimensionalen “Körper” als Felder zu bezeichnen.
    Nur das 1-dimensionale Feld wird häufig auch als **Vektor** *oder* ’**array**’, das 2-dimensionale Feld auch als **Matrix** bezeichnet.
  - die Wertebereiche der Indexgruppen bestimmen die **Gestalt** eines Feldes - etwa: gleiche Wertebereiche ergeben Quadrate, Würfel; ungleiche Wertebereiche Rechtecke bzw. Quader.
  - die **Größe** eines Feldes ist die Summe der Wertebereiche der Indexgruppen (multipliziert mit der Länge der Einzelelemente) also der Speicherbedarf eines Feldes.
    Intern werden alle mehrdimensionale Felder als 1-dimensionale Vektoren abgespeichert, da Arbeitsspeicher 1-dimensional organisiert sind.

Feldverarbeitung

■ Feldarten

▲ Felder mit expliziter Gestalt

- ug kann angegeben werden; default: 1
- og muß angegeben werden

```
Syntax: ([ug:]og[,[ug:]og]...)
```

Beispiel:

```
SUBROUTINE manip(feld_a, max)
!Formalparameter:
INTEGER max
REAL, DIMENSION (-10:max,10,max) :: feld_a
!Lokale Variable:
INTEGER, DIMENSION (10,20,10) local_array
.
END SUBROUNTINE manip
```

Indexgrenzen, die nichtkonstante Spezifikationsausdrücke sind, dürfen nur in der Feldvereinbarung von Formalparametern oder von automatischen Feldern (siehe später in diesem Kapitel) auftreten.

Feldverarbeitung

- Felder mit übernommener Gestalt
  - sind **Formalparameterfelder** (ohne POINTER-Attribut!), die ihre Gestalt von den zugeordneten Aktualparameterfeldern übernehmen; also die Wertebereiche der Indexgruppen.
  - Bei der Typvereinbarung des Formalparameterfeldes wird nur der Rang des Feldes durch die Angabe von ":" spezifiziert:
    - ug kann angegeben werden; default: 1
    - og **darf nicht** angegeben werden
  - man kann also die Wertebereiche der Indizes verschieben

```
Syntax: ([ug]:[,[ug]:]...)
```


Feldverarbeitung

- Felder mit übernommener Gestalt (cont)

Beispiel:

```
PROGRAM demo
REAL, DIMENSION (30,30,-15:14) :: feld_org
.
CALL manip(feld_org)
.
CONTAINS
SUBROUTINE manip(feld_formal)
!Formalparameter:
REAL, DIMENSION(-15:,-15:,:) :: feld_formal
!Lokale Variable:
INTEGER, DIMENSION (10,20,10) local_array
.
.
END SUBROUNTINE manip
END PROGRAM demo
```

Feldverarbeitung

- Felder mit übernommener Größe
  - sind **Formalparameterfelder** (ohne POINTER-Attribut!), die ihre **Größe** von den zugeordneten Aktualparameterfeldern übernehmen; also nicht den Rang!
  - Hiermit hat man die Möglichkeit Felder verschiedener Dimensionen aufeinander abzubilden, wobei der “Größenabgleich” über die letzte Dimension des Formalparameterfeldes vorgenommen wird.

Von dieser Möglichkeit sollte nur dann Gebrauch gemacht, wenn es ein Algorithmus auch “erfordert”. Ein Kommentar über das Warum sollte niemals fehlen, da sonst die Durchsichtigkeit des Programms leidet.

- Solche Anpassungen könnten z. B. dann notwendig sein, wenn man vordefinierte (auch eigene) Funktionen verwenden möchte, die eine vorgegebene Struktur eines Feldes erzwingen.

Feldverarbeitung

- Felder mit übernommener Größe (cont)
  - ug kann angegeben werden; default: 1
  - als og der letzten Dimension muß ein “*” angegeben werden

```
Syntax: ([ug:]og,]...[ug:] * )
```

Beispiel:

```
PROGRAM demo
REAL, DIMENSION (30,30,-15:14) :: feld_org
.
CALL manip (feld_org)
.
CONTAINS
SUBROUTINE manip(feld_formal)
!Formalparameter:
REAL, DIMENSION(30,-30:*) :: feld_formal
!Lokale Variable:
INTEGER, DIMENSION (10,20,10) local_array
.
.
END SUBROUTINE manip
END PROGRAM demo
```

Feldverarbeitung

Feldorganisation

- Die interne Struktur eines Feldes
  - ist 1-dimensional.
  - Höherdimensionale Felder werden auf 1-dimensionale Vektoren spaltenweise abgebildet, d.h. der 1. Index “läuft” am schnellsten der letzte am langsamsten.
  - Beispiel: feld (4,10)

feld (4,10)

1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10
2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10
3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10

Speichervektor

1,1	2,1	3,1	4,1	1,2	2,2	3,2	4,2	1,3	2,3	3,3	3,4	1,4	2,4	3,4	4,4	1.5	...	...	4,10

Feldverarbeitung

Feldorganisation(cont)

- Beispiel: feld(3,3,4)

Ebene 1

feld(:,:,1)

1,1,1	1,2,1	1,3,1
2,1,1	2,2,1	2,3,1

Ebene 2

feld(:,:,2)

1,1,2	1,2,2	1,3,2
2,1,2	2,2,2	2,3,2

Ebene 3

feld(:,:,3)

1,1,3	1,2,3	1,3,3
2,1,3	2,2,3	2,3,3

Ebene 4

feld(:,:,4)

1,1,4	1,2,4	1,3,4
2,1,4	2,2,4	2,3,4

Speichervektor

1,1,1	2,1,1	3,1,1	1,2,1	2,2,1	3,2,1	1,3,1	2,3,1	3,3,1	1,1,2	2,1,2	3,1,2	1,2,2	2,2,2	...	...	3,3,4

- Formel zur Berechnung einer Position

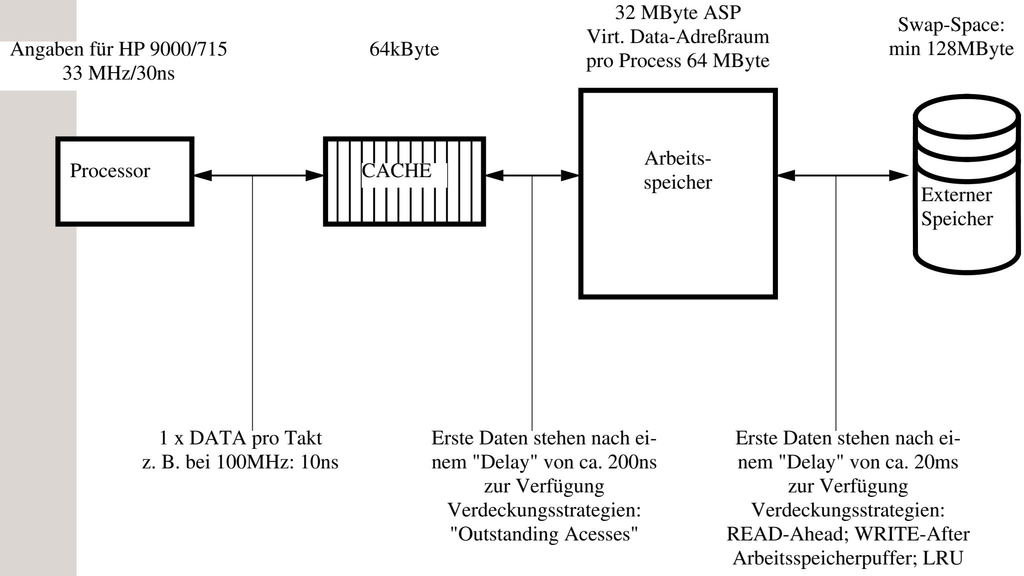
$$1 + \langle a_1 - ug_1 \rangle + \sum_{m=1}^{n-1} ((ai_{m+1} - ug_{m+1}) \bullet \prod_{i=1}^{m} (og_i - ug_i + 1))$$

a_i = aktueller Index der Variablen der i-ten Dimension
ug_i = untere Grenze der i-ten Dimension
og_i = obere Grenze der i-ten Dimension
n = Rang des Feldes

Feldverarbeitung

■ Feldorganisation(cont)

- Rechnerarchitekturabhängigkeiten



Feldverarbeitung

■ Feldorganisation(cont)

▲ warum ist das Wissen um diese Gegebenheiten so bedeutend?

- Abarbeitungsfolgen (in Schleifen) sollten “lokal” gehalten werden, wegen:
  - der Berechnungskomplexität der Zieladressen
    (Formel Seite 9 oder (befehlsparalleles) Increment)
  - der Cache-Effekte
  - der Paging-Effekte
  - Speicherinterleaving - Effekte (memstride)

```
PROGRAM thrash
INTEGER, PARAMETER :: max = 64
INTEGER, DIMENSION (max,max,max) :: feld
INTEGER i,j,k
DO i = 1,max
   Do j = 1,max
      DO k = 1,max
            feld(i,j,k) = 1
      END DO
   END DO
END DO
END PROGRAM trash
```

Feldverarbeitung

- Feldorganisation(cont)

Aufgabe:

Messen Sie mit dem time-Kommando die Laufzeiten des Programms und variieren Sie die Reihenfolge der Indizes und die Gestalt des Feldes. Bei welchen Größen treten die “Einbrüche” bei der HP 9000/715 besonders deutlich auf und warum?
Damit die Werte vergleichbar bleiben, müssen Sie durch Anpassung des Programms dafür Sorge tragen, daß die dynamische Anzahl der ausgeführten Operationen konstant bleibt.

- Benutzung des time-Kommandos

```
time filename.out
```

- Ergebnis:

```
d.ddu (usertime) d.dds (systemtime) d:dd.d (Verweilzeit) &
d.dd% (CPU-Nutzung)
z.B. 0.11u 3.97s 0:04.17 97.8%
(Beispiel mit DEALLOCATE S. 31/32 auf Sun SparcStation 20)
```


Feldverarbeitung

- Zugriffe auf Felder (Array-Syntax Fortran90/95)
  - Zugriffe auf Felder lassen sich in Zugriffe auf
    - Feldelemente
    - ganze Felder und
    - Teilfelder gliedern.
  - Zugriff auf Einzelelemente
    - auf Feldelemente eines Feldes

Beispiel:

```
REAL, DIMENSION (10,20,10) :: feld
DATA feld /2000 * 1./! das gesamte Feld wird statisch
                     ! mit 1.0 vorbesetzt

feld(9,15,4) = 0.0   ! Zugriff auf das 949.
                     ! Speicherelement
                     ! nach Formel Seite 9; Nachrechnen!
```

Feldverarbeitung

- Zugriffe auf Felder
  - Zugriff auf Einzelelemente (cont)
    - auf Feldelemente einer Strukturkomponente

```
Beispiel:
TYPE datum
   INTEGER tag
   CHARCTER *3 monat
   INTEGER, DIMENSION (24) :: stunde
END TYPE datum

TYPE datum, DIMENSION (365) :: kalender

kalender(8)                 ! eine Struktur als Feldelement
kalender(8)%tag             ! eine Strukturkomponente; kein
                            ! Feldelement
kalender(5)%stunde(12)      ! ein Feldelement einer
                            ! Strukturkomponente
```


Feldverarbeitung

- Zugriff auf ganze Felder
  - wird nur der Feldname als Variable genannt, so ist damit das Gesamtfeld (genauer: die Anfangsadresse des Feldes) gemeint.

```
Beispiele:
REAL, DIMENSION (10,20) :: feld
DATA feld /200 * 1./ ! das gesamte Feld wird statisch
                     ! mit 1.0 vorbesetzt

feld = 0.0           ! Allen Feldelementen des Feldes
                     ! wird der Wert 0.0 zugewiesen
```

Feldverarbeitung

- Zugriffe auf Teilfelder

werden durch Angabe der Wertefolgen der Indizes spezifiziert und zwar durch

- einen Indexwert
- ein Indextripel:

```
(aw : ew : ia)
aw = Anfangswert; ew = endwert; ia = indexabstand(default: 1)
```

- einen Vektorindex = ein eindimensionaler Feldausdruck der Form:

```
PARAMETER ( i = (/ ic, ic, ic,.../) ! ic = Integerkonstante
```

Mindestens einer der Teilfeldindizes muß ein Indextripel oder ein Vektorindex sein.

Feldverarbeitung

- Zugriffe auf Teilfelder (cont)
  - Indextripel:

```
REAL, DIMENSION (-10,89) :: f

f(5:12)          ! f(5),f(6) ...f(12)
f(-8:20:4)       ! f(-8),f(-4),f(0),f(4) ...f(20)
f(10:0:-1)       ! f(10), f(9), ...f(0)
```

- Indexwert und Indextripel:

```
INTEGER, DIMENSION (4,5,10) :: c

c(2,2,:)         ! ist ein eindimensionales Teilfeld der
                 ! Größe 10
c(2,3,2:10:2)    ! c(2,3,2),c(2,3,4), ....c(2,3,10)
c(:,5,:)         ! Ein 2-dimensionales Teilfeld
                 ! der Dimension 4 x 10
c(:,2:5,7:10)    ! Ein 3-dimensionales Teileld
                 ! der Dimension 4 x 4 x 4
```

Feldverarbeitung

- Zugriffe auf Teilfelder (cont)
  - Indexwerte und Indexvektoren

```
INTEGER, DIMENSION(5) :: v1, v2
DATA v1 = /1,3,5,7,9/
DATA v2 = /9,7,5,3,1/
f(5,v1)         ! f(5,1),f(5,3),f(5,5),...f(5,9)
f(v2,3)         ! f(9,3),f(7,3),f(5,3),...f(1,3)
f(v1,v2)        ! f(1,9),f(3,7),f(5,5),...f(9,1)
```

  - Indexwerte, Indextripel und Indexvektoren

```
REAL, DIMENSION (10,10,10) :: f
INTEGER, DIMENSION (3) :: v
DATA v = /1,9,3/
f(v,5,1:10:4)  ! f(1,5,1),f(9,5,5),f(3,5,9)
```

  - Auch diese Konstruktionen sind nur dann zu verwenden, wenn es die Berechnugsstrukturen erfordern - also vereinfachen bzw. effizienter gestalten! Kommentieren!

Feldverarbeitung

- Feldkonstanten, Konstruierte Feldwerte haben wir bereits bei anderen Gelegenheiten kennengelernt; hier zur Erinnerung einige Beispiele:

```
Feldkonstante:

CHARACTER *8, DIMENSION (5) :: vorname
INTEGER, DIMENSION (10) :: feld1
PARAMETER (feld1 = (/0,1,2,3,4,5,6,7,8,9/)
.
.
vorname = (/'Paulchen',"Annalena', 'Claus-Uwe'/)
feld1 = (/(i, i=0,8,2), (i, i=1,9,2)/)


Konstruierte Feldwerte:

REAL, DIMENSION (20) :: feld1, feld2

feld1=(/(0.0, i=1,10),feld2(1:4) * feld2(11:14),(i, i=6,1,-1)/)
```

Feldverarbeitung

- Operationen, Zuweisungen auf Felder
  - die uns bekannten (vordefinierten) Operatoren wirken komponentenweise auf Feldausdrücke, wenn diese konform sind.
  - Zwei Felder werden als **konform** bezeichnet, wenn sie in der Gestalt gleich sind, **d.h. den gleichen Rang und in allen Dimensionen die gleiche Größe besitzen.**
  - Stehen bei Zuweisungen auf der rechten Seite Felder bzw. Teilfelder, so erfolgt die Zuweisung komponentenweise, wenn die Felder auf beiden Seiten konform sind.
  - Steht bei einer Zuweisung auf der rechten Seite ein skalarer Ausdruck, dann wird allen Elemente der linken Seite der skalare Wert zugewiesen.

Feldverarbeitung

- Operationen, Zuweisungen auf Felder (cont)

Beispiele:

```
REAL feld1(5,5), feld2(5,10), feld3(10,10,10)
REAL, DIMENSION (10,10) :: x,y,z
.
.
x = y ** z
z = feld3(1,:,:) + feld3(:,:,3)
feld1 = 1.0
feld2(:,1:10:2)= feld1 + x(1:5,1:5)
```

Feldverarbeitung

- Operationen, Zuweisungen auf Felder (cont)

Neben der Möglichkeit der vordefinierten Operatoren gibt es noch eine Anzahl vordefinierter Standard-Funktionen (Siehe auch RRZN-Handbuch S. 323ff) für die Feldverarbeitung wie:

- MATMUL
  Matrizenmultiplikation, Vektor-Matrix-Multiplikation
- DOTPRODUCT
  Skalarprodukt
- ALLOCATED, SIZE, SHAPE, LBOUND, UBOUND
  Ermittelung der Eigenschaften von Feldern
- MERGE, SPREAD, RESHAPE, PACK, UNPACK
  Konstruktion von Feldern
- SUM, PRODUCT, MAXVAL, MINVAL, COUNT, ANY, ALL
  Berechnungen mit Rangreduktionen (-transformationen)

3.1.8 Dynamische Speicherverwaltung

- Ein **automatisches Feld**
  - ist ein Feld, daß im Geltungsbereich eines Unterprogramms vereinbart ist und **kein** Formalparameterfeld ist - also ein lokales Feld.
  - Das Feld ist von expliziter Gestalt, wobei mindestens **ein Index eine Variable sein muß.**
  - die Lebensdauer des Feldes ist - wie bei anderen lokalen Variablen auch - auf die Dauer **eines** Aufrufs des Unterprogramms beschränkt.
  - Eine “Static-Variable” mit Hilfe einer SAVE-Anweisung bzw. SAVE-Attributs kann ein automatisches Feld nicht werden.
  - Der benötigte Speicherplatz wird zum Aufrufzeitpunkt dynamisch angelegt und zum Rücksprungzeitpunkt wieder freigegeben.

Dynamische Speicherverwaltung

- Automatische Felder (cont)

Beispiel:

```
SUBROUTINE vektortausch (f1, f2, vn, breite, laenge)

! Effekt: Vertauscht jeweils einen Vektor der 2-dimensionalen
! Felder f1 und f2
! Parameter: vn = Vektornummer; breite, laenge Dim. der Felder
! Formalparameter:

REAL, DIMENSION (breite, laenge) :: f1, f2
INTEGER vn, breite, laenge
! Lokale Groessen:

REAL, DIMENSION (laenge) :: hilfsfeld

   hilfsfeld = f1(vn, :)
   f1(vn, :) = f2(vn, :)
   f2(vn,:) = hilfsfeld

END SUBROUTINE vektortausch
```


Dynamische Speicherverwaltung

- Dynamische Felder
  - werden dann verwendet
    - wenn erst zur Laufzeit bekannt ist, wie groß der benötigte Speicherplatz für die Daten sein wird oder
    - wenn die verwendeten Datenbereiche so groß werden, daß der für eine Anwendung verfügbare (virtuelle) Speicher nicht ausreicht; man also selbst einen “overlay” erzeugen **muß** oder
    - wenn der physikalische Arbeitsspeicher nicht ausreicht und es aufgrund der Kenntnis der Datenstrukturen für die Berechnung effizienter ist, die Speicherverwaltung selbst zu steuern, als es dem Betriebssystem (Paging) zu überlassen.
    - Bemerkung: Klassische Vektorrechner haben in der Regel weder Cashes noch eine virtuelle Speicherverwaltung. Hier ist ein entscheidendes Geschwindigkeitskriterium die Länge des Vektors, die mit einem Takt verarbeitet werden kann. Die Speicherverwaltung ist physikalisch, bei großen Programm sind also ’overlays’ notwendig.

Dynamische Speicherverwaltung

- Dynamische Felder
  - sind Felder, deren Typ und Rang (durch die Anzahl der Doppelpunkte) vereinbart wird, nicht aber die Größe und Gestalt. Die ug und og dürfen also nicht spezifiziert sein.

    `Syntax: (:[,:]...)`
  - Dynamische Felder haben das Attribute ALLOCATABLE.
  - Für die Geltungsbereiche gilt das gleiche wie für normale Variable.
  - Formalparameterfelder können keine dynamischen Felder sein
  - Der Speicherbereich wird mit der ALLOCATE-Anweisung angefordert und mit der DEALLOCATE-Anweisung wieder freigegeben, letzteres erfolgt auch bei der Ausführung einer RETURN- oder END-Anweisung.
  - Analog gilt für die Lebensdauer, daß sie mit der ALLOCATE-Anweisung beginnt und mit der DEALLOCATE-, RETURN- oder END-Anweisung endet.

Dynamische Speicherverwaltung

- Dynamische Felder
  - Anfordern, Freigeben eines Speicherbereichs

```
Syntax:
ALLOCATE (feld(dim[,dim]...)[,feld(dim[,dim]...)]...
[,STAT=statvar])
DEALLOCATE (feld[,feld]...[,STAT=statvar])
```

(dim[,dim]...) entspricht der Typvereinbarung eines Feldes mit expliziter Gestalt. Das Laufzeitsystem berechnet den aktuellen Speicherbedarf und fordert diesen beim Betriebssystem an.
Betriebssysteme liefern bei jedem Aufruf (“system-call”) einen Fehler- bzw. Statuscode zurück. Das Laufzeitsystem von Fortran90 liefert diesen Parameter weiter an den Benutzer, indem es den Code in der angegeben Statusvariablen hinterlegt.

Es ist ein ausgesprochen schlechter (und verbreiteter) Programmierstil die Statusvariablen nicht auszuwerten!

Dynamische Speicherverwaltung

- Dynamische Felder (cont)
  - Vereinbarung, Anfordern, Freigeben

```
REAL, DIMENSION (:,:), ALLOCATABLE :: a
INTEGER, ALLOCATABLE :: b(:,:,:)
INTEGER min, max, hoehe, breite, laenge
INTEGER statusvar
.
.
ALLOCATE(a(min:100,100:max), b(hoehe, breite, laenge),&
       STAT=statusvar)
IF (statusvar > 0) CALL exception ('ALLOCATE',statusvar)
.
.
.
DEALLOACATE (a,b, STAT=statusvar)
IF (statusvar > 0) CALL exception ('DEALLOCATE',statusvar)
.
```


Dynamische Speicherverwaltung

- Dynamische Felder mit Zeiger-(Pointer-) Variable
  - Wie wir bereits im Kapitel 3.1.7 erfahren haben, können Pointer auch auf Felder verweisen; also die Anfangsadresse eines Feldes zum Inhalt haben.
    Dies geschieht analog zu den Zugriffen auf ganze Felder oder Teilfelder. Auch hier wird der Name des Feldes als Anfangsadresse des Feldes interpretiert. Es liegt also nahe, dynamische Felder auch mit sogenannten **Feldzeigern** zu verwalten.
  - Die Verarbeitung erfolgt völlig analog zum bisher Beschriebenen, nur bei der Typvereinbarung wird das POINTER-Attribut angegeben; **nicht** zusätzlich auch noch das ALLOCATABLE-Attribut.
  - Das durch eine ALLOCATE-Anweisung generierte Feld erhält automatisch implizit des TARGET-Attribut.
  - Mit Hilfe des Pointers lassen sich auch Datenstrukturen (selbstdef. Typen) mit Hilfe der dynamischen Speicherverwaltung verarbeiten (siehe Beispiele).

Dynamische Speicherverwaltung

- Dynamische Felder mit Zeiger-(Pointer-) Variable (cont)
  - Vereinbarung, Anfordern, Freigeben

```
REAL, DIMENSION (:,:), POINTER :: a! Zeiger auf ein
                                ! 2-dimensionales Feld
INTEGER, POINTER :: b(:,:,:)    ! zeiger auf ein
                                ! 3-dimensionales Feld
INTEGER min, max, hoehe, breite, laenge
INTEGER statusvar
.
.
ALLOCATE(a(min:100,100:max), b(hoehe, breite, laenge),&
       STAT=statusvar)
IF (statusvar > 0) CALL exception ('ALLOCATE',statusvar)
.
.
.
DEALLOACATE (a, b, STAT=statusvar)
IF (statusvar > 0) CALL exception ('DEALLOCATE',statusvar)
.
```


Dynamische Speicherverwaltung

- Fehlerbehandlung am Beispiel

```
PROGRAM exceptdemo

INTEGER i, iostat
INTEGER, DIMENSION (:,:), ALLOCATABLE :: feld

DO i = 10, 10000, 10

   ALLOCATE (feld(i,i*10), STAT=iostat)
   PRINT *,i, i*10, iostat
   IF (iostat > 0) CALL exception ('ALLOC', iostat)
!  DEALLOCATE (feld, STAT=iostat)
!  IF (iostat > 0) CALL exception ('DEALL', iostat)

END DO

CONTAINS

SUBROUTINE exception (syscall, iostat)
CHARACTER *5 syscall
INTEGER iostat
```

Dynamische Speicherverwaltung

```
      SELECT CASE (syscall)
            CASE ('ALLOC')
                  PRINT *, syscall, ': ERRCODE: ', iostat
                  IF (iostat == 1) PRINT *, 'NO MORE MEMORY AVAIL.'
                  IF (iostat == 2) PRINT *, 'VAR. ALREADY ALLOCATED
            CASE ('DEALL')
                  PRINT *, syscall, ': ERRCODE: ', iostat
            CASE ('WRITE')
                  PRINT *, 'Not yet implemented'
            CASE ('READ ')
                  PRINT *, 'Not yet implemented'
            CASE ('REWND')
                  PRINT *, 'Not yet implemented'
            CASE DEFAULT
                  PRINT *, 'Unbekannte Fehlerursache'
            END SELECT
      STOP 'panic'

END SUBROUTINE exception
END PROGRAM exceptdemo
```


3.1.9 Bedingte Feldzuweisungen

- WHERE / ELSEWHERE / END WHERE

```
IF (b < 0.0) b = 0.0
```

Will man eine solche Operation auf eine Anzahl von Feldelementen ausführen mußten wir bisher eine Schleife konstruieren:

```
...
DO i = 1,n
   IF (array(i) < 0.0) array(i) = 0.0
END DO
...
```

analog wirkt die WHERE-ANWEISUNG

```
...
WHERE (array < 0.0) array = 0.0
...
```

Bedingte Feldzuweisungen

- Allgemeine Form der WHERE - Anweisung

```
Syntax:WHERE (logischer maskenausdr.) feldvar =
feldausdr.
```

logischer Maskenausdruck und Feldvariable in der Feldzuweisung müssen von gleicher Gestalt sein!

- Allgemeine Form der WHERE - Konstrukts

```
Syntax:WHERE (logischer maskenausdr.)
      [feldanweisungen]
      ...
[ELSEWHERE
      [feldanweisungen]
      ...]
END WHERE
```

logischer Maskenausdruck und Feldvariable in der Feldzuweisung müssen von gleicher Gestalt sein!

Bedingte Feldzuweisungen

```
Beispiel:
PROGRAM where
IMPLICIT NONE
REAL, DIMENSION(5,5) :: felda, feldb
INTEGER :: i,j

! Maske ueber Felder legen
LOGICAL, DIMENSION (5,5) :: maske = .FALSE.
   DO i = 1,5
       DO j = 1,5
      IF (i == j) maske (i,j) = .TRUE.
       END DO
   END DO
   WHERE (maske)
       felda = 1.00
   ELSEWHERE
       felda = 5.00
   END WHERE
   PRINT *,' '
   PRINT *, 'Diagonale mit 1.0 besetzt; Rest 5.0'
   PRINT *,' '
   DO i = 1,5
       PRINT *, felda(i,:)
   END DO
```

Bedingte Feldzuweisungen

```
! Ueber logische Bedingung auf Einzelelmente agieren

   feldb = 6.0
   WHERE (felda > 1.0)
       feldb = felda * feldb
   ELSEWHERE
       feldb = 4.0
   END WHERE

   PRINT *,' '
   PRINT *, 'Diagonale mit 4.0 besetzen; Rest felda * feldb'
   PRINT *,' '

   DO i = 1,5
       PRINT *, feldb(i,:)
   END DO

END PROGRAM where
```