Anweisungen in Fortran - Unterprogramme

Unterprogramme und Funktionen

Vereinbarung:

`SUBROUTINE name [(fpar)]`

Vereinbarung formaler Parameter
und interner Variablen
Anweisungen ...

```
RETURN
END
```

`typ FUNCTION fname([fpar])`

Vereinbarung formaler Parameter
und interner Variablen
Anweisungen ...

```
fname = Funktionswert
RETURN
END
```

Aufruf:

`CALL name [(akt_par)]`

`f = fname([akt_par])`

`fpar`	Liste formaler Parameter, Variablennamen gelten (nur) innerhalb des Unterprogramms
`akt_par`	Liste aktueller Parameter, (call-by-reference) in angegebener Reihenfolge den formalen Parametern zugewiesen

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle INTENT

- Vereinbarungen der formalen Parameter müssen mit denen der aktuellen Parameter übereinstimmen – anderenfalls: „auf eigene Gefahr“ mit Fehlinterpretationen rechnen, denn ...
- Parameterübergabe prinzipiell „by reference“, d.h. das Unterprogramm „kennt“ die Speicheradressen der aktuellen Parameter; Wertzuweisung an formalen Parameter ändert aktuellen Parameter im rufenden Programm
- Das gilt auch für FUNCTION-Unterprogramme, ist hier aber ein sog. „Seiteneffekt“, weil Funktionen nur **einen** Wert (über den Funktionsnamen) zurückgeben sollten.
- F90 bietet Absicherung gegen versehentlich falsche Verwendung formaler Parameter, Attribut `INTENT(...)`, mit einem der Argumente `IN`, `OUT` oder `INOUT`, (als Hinweis an den Compiler, Optimierung, Warnungen) z. B.

```
SUBROUTINE make_matrix(n,A)
INTEGER, INTENT(IN) ::  n
REAL, DIMENSION(n,n), INTENT(OUT) ::  A
```

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Parameterübergabe

- Ans UP wird nur Adresse des aktuellen Parameters übergeben. Interpretation des Speicherinhalts richtet sich ausschließlich nach Vereinbarungen im UP.
- Skalare Parameter werden aus Effizienzgründen im UP temporär zwischengespeichert, bei Feldern wird auf Originaladressen zugegriffen. Bei nichtkorrekter Vereinbarung mögliche Fehler beim „Durchreichen“ von Parametern an weitere UP:

PROGRAM HP

```
DIMENSION A(10)
n=10
call UP1(n,A)
...
END
```

SUBROUTINE UP1(n,A)

```
INTEGER n
REAL A
...
call UP2(n,A)
...
END
```

SUBROUTINE UP2(n,A)

```
INTEGER n,i
REAL A(n)
DO i=1,n
  A(i)=i
ENDDO
END
```

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Parameterübergabe

- Ans UP wird nur Adresse des aktuellen Parameters übergeben. Interpretation des Speicherinhalts richtet sich ausschließlich nach Vereinbarungen im UP.
- Skalare Parameter werden aus Effizienzgründen im UP temporär zwischengespeichert, bei Feldern wird auf Originaladressen zugegriffen. Bei nichtkorrekter Vereinbarung mögliche Fehler beim „Durchreichen“ von Parametern an weitere UP:

PROGRAM HP

```
DIMENSION A(10)
n=10
call UP1(n,A)
...
END
```

SUBROUTINE UP1(n,A)

```
INTEGER n
REAL A      ! Fehler
...
call UP2(n,A)
...
END
```

SUBROUTINE UP2(n,A)

```
INTEGER n,i
REAL A(n)
DO i=1,n
  A(i)=i
ENDDO
END
```

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Parameterübergabe

- Ist der aktuelle Parameter ein Ausdruck, wird dessen Wert temporär als Hilfsvariable gespeichert und deren Adresse ans UP übergeben.
  Beachte: Typ des Ausdrucks ist von Operanden abhängig.
- Bei Zeichenketten als Parameter muss das UP nicht die (konstante) Länge der Zeichenkette vereinbaren, sondern z. B.:

```
SUBROUTINE Fehlermeldung(Text)
CHARACTER*(*) Text
IF (LEN(Text) .GT. 80) THEN
  ...
```

  Die Länge der aktuell übergebenen Zeichenkette wird in der Rufzeile als (unsichtbarer) zusätzlicher Parameter ans Ende der Liste angefügt, aber nicht „by-reference“, sondern „by-value“.
  Dasselbe Unterprogramm in C wäre also bspw.:

```
void fehlermeldung_(char* Text, int L) {
  if (L > 80) { ...
```

Unterprogramm-Schnittstelle — Parameterübergabe

- Bis F77 waren Dimensionsangaben für Felder grundsätzlich Konstante (statische Vereinbarung des Speicherbedarfs zur Übersetzungszeit).
- Variable als Felddimension für formale Parameter von Unterprogrammen sind nur erlaubt, wenn diese variable Dimension selbst als Parameter in der Rufzeile oder als Variable in einem Common-Block ans UP übergeben wird.
- Startadresse eines 1D-Feldes und Index i reichen aus, um die aktuelle Speicherposition des i-ten Elements zu bestimmen. Somit muss die wahre Länge des Feldes im UP nicht explizit vereinbart sein,
  es genügt ein Platzhalter (`*`), z. B.

```
SUBROUTINE mult(n,x,y,z)
INTEGER n
DIMENSION x(*),y(*),z(*)
```

  auch möglich: `DIMENSION x(n),y(n),z(n)`
  oder (früher üblich): `DIMENSION x(1),y(1),z(1)`

Unterprogramm-Schnittstelle — Parameterübergabe

- Bis F77 waren Dimensionsangaben für Felder grundsätzlich Konstante (statische Vereinbarung des Speicherbedarfs zur Übersetzungszeit).
- Variable als Felddimension für formale Parameter von Unterprogrammen sind nur erlaubt, wenn diese variable Dimension selbst als Parameter in der Rufzeile oder als Variable in einem Common-Block ans UP übergeben wird.
- Startadresse eines 1D-Feldes und I… Speicherposition des *i*-ten Element… Länge des Feldes im UP nicht expl… es genügt ein Platzhalter (`*`), z. B.

```
SUBROUTINE mult(n,x,y,z)
INTEGER n
DIMENSION x(*),y(*),z(*)
```

auch möglich: `DIMENSION x(n),y(n),z(n)`
oder (früher üblich): `DIMENSION x(1),y(1),z(1)`

Bei mehrdimensionalen Feldern darf nur die letzte Dimensionsangabe durch einen Platzhalter ersetzt werden:

```
REAL A(n,*), B(n,1), C(n,m,*)
```

`A(i,k)` $\rightarrow$ `(k-1)*n+i`
`C(i,j,k)` $\rightarrow$ `(k-1)*n*m+(j-1)*n+i`

Zwischenbemerkung - Speicherabbildungsfunktion

Mehrdimensionales Array: `A(i1:k1,...,in:kn)`

Berechnung der Position des Elements `A(j1,...,jn)`:

$$S = [1+] \sum_{m=1}^{n} (j_m - i_m) \cdot D_m$$

D_m: Entfernung zweier Elemente im Speicher, die sich im Index j_m um 1 unterscheiden

spaltenweise

$$\begin{aligned} D_1 &= 1 \\ D_{m+1} &= (k_m - i_m + 1) \cdot D_m \end{aligned}$$

zeilenweise

$$\begin{aligned} D_n &= 1 \\ D_{m-1} &= (k_m - i_m + 1) \cdot D_m \end{aligned}$$

Durch Verwendung von Dope-Vektoren zur internen Definition eines Arrays (z. B. in der Sprache PL/1) kann man auf Kopien von Submatrizen verzichten.

Dope-Vektor-Inhalt:

virtuelle Basisadresse `C` des Elements `A(0,...,0)`, sowie alle D_m, k_m, i_m.

Abbildung: `A(j1,...,jm)` $\rightarrow C + \sum_{m=1}^{n} j_m D_m$.

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Parameterübergabe

Zwei Möglichkeiten zur Übergabe von Teilfeldern (Submatrizen) als aktuelle Parameter im HP: `DIMENSION A(100,500)`, dann als Submatrix Zeilen $i_1 \ldots i_2$, Spalten $j_1 \ldots j_2$, wobei: `n=i2-i1+1` und `m=j2-j1+1` .

elegant + speicher-ineffizient

```
call up(n,m,A(i1:i2,j1:j2))
```

```
subroutine up(n,m,A)
dimension A(n,m)
```

temporäre, dicht gepeicherte Kopie der Submatrix als akt. Parameter

unelegant + speicher-effizient

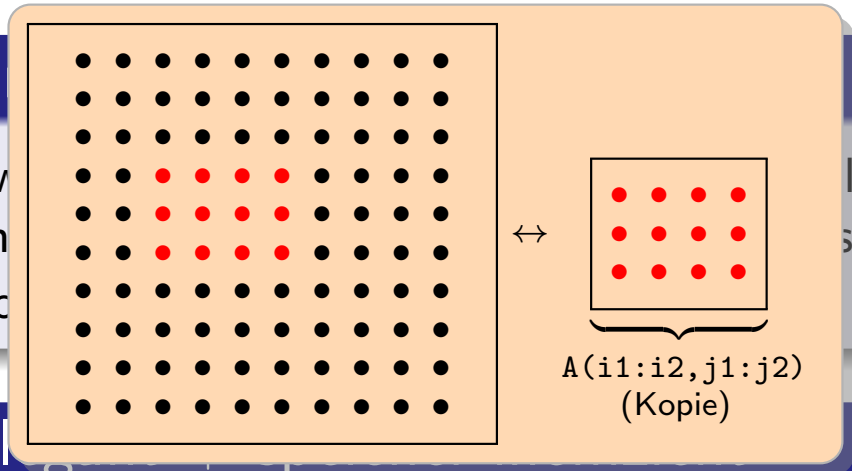
```
call up(n,m,A(i1,j1),100)
```

```
subroutine up(n,m,A,LDA)
dimension A(LDA,*)
```

Originalspeicherplatz, größere Lücken zwischen Spalten (jeweils LDA$-n$)

Verarbeitung in beiden UP-Varianten jeweils: `A(i,j)` für `i=1,n` und `j=1,m`

Anweisungen in Fortran - Unterprogramme



U… Parameterübergabe

Zv…lfeldern (Submatrizen) als aktuelle Parameter
im… Submatrix Zeilen $i_1 \ldots i_2$, Spalten $j_1 \ldots j_2$,
wo…

el…

```
call up(n,m,A(i1:i2,j1:j2))
```

```
subroutine up(n,m,A)
dimension A(n,m)
```

temporäre, dicht gepeicherte Kopie
der Submatrix als akt. Parameter

unelegant + speicher-effizient

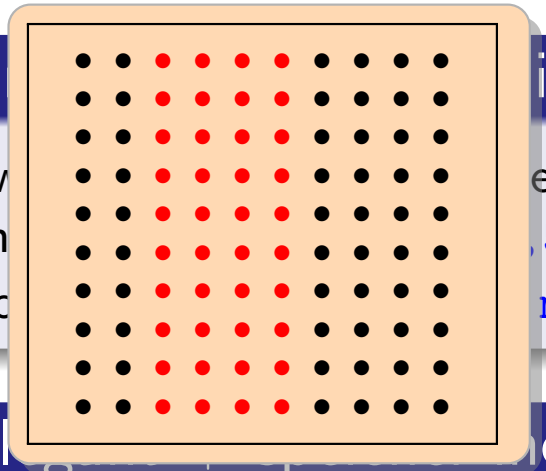
```
call up(n,m,A(i1,j1),100)
```

```
subroutine up(n,m,A,LDA)
dimension A(LDA,*)
```

Originalspeicherplatz, größere Lücken
zwischen Spalten (jeweils LDA$-n$)

Verarbeitung in beiden UP-Varianten jeweils: `A(i,j)` für `i=1,n` und `j=1,m`

Anweisungen in Fortran - Unterprogramme



U ittstelle Parameterübergabe

Zv ergabe von Teilfeldern (Submatrizen) als aktuelle Parameter im ,500), dann als Submatrix Zeilen $i_1 \ldots i_2$, Spalten $j_1 \ldots j_2$, wo m=j2-j1+1 .

el effizient

```
call up(n,m,A(i1:i2,j1:j2))
```

```
subroutine up(n,m,A)
dimension A(n,m)
```

temporäre, dicht gepeicherte Kopie der Submatrix als akt. Parameter

unelegant + speicher-effizient

```
call up(n,m,A(i1,j1),100)
```

```
subroutine up(n,m,A,LDA)
dimension A(LDA,*)
```

Ohne Kopie, wenn n = volle Spaltenlänge, also etwa: `call up(n,m,A(:,j1:j2))` ($i_1 = 1, i_2 = 100 = n$)

Verarbeitung in beiden UP-Varianten jeweils: `A(i,j)` für `i=1,n` und `j=1,m`

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Parameterübergabe

Zwei Möglichkeiten zur Übergabe von Teilfeldern (Submatrizen) als aktuelle Parameter im HP: `DIMENSION A(100,500)`, dann als Submatrix Zeilen $i_1 \ldots i_2$, Spalten $j_1 \ldots j_2$, wobei: `n=i2-i1+1` und `m=j2-j1+1` .

elegant + speicher-ineffizient

```
call up(n,m,A(i1:i2,j1:j2))
```

```
subroutine up(n,m,A)
dimension A(n,m)
```

temporäre, dicht gepeicherte Kopie der Submatrix als akt. Parameter

unelegant + speicher-effizient

```
call up(n,m,A(i1,j1),100)
```

```
subroutine up(n,m,A,LDA)
dimension A(LDA,*)
```

Originalspeicherplatz, größere Lücken zwischen Spalten (jeweils `LDA`$-n$)

Verarbeitung in beiden UP-Varianten jeweils: `A(i,j)` für `i=1,n` und `j=1,m`

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Parameterübergabe

Zwei Möglichkeiten zur Übergabe von Teilfeldern (Submatrizen) als aktuelle Parameter im HP: `DIMENSION A(100,500)`, dann als Submatrix Zeilen $i_1 \ldots i_2$, Spalten $j_1 \ldots j_2$, wobei: `n=i2-i1+1` und `m=j2-j1+1` .

elegant + speicher-ineffizient

```
call up(n,m,A(i1:i2,j1:j2))
```

```
subroutine up(n,m,A)
dimension A(n,m)
```

temporäre, dicht gepeicherte Kopie der Submatrix als akt. Parameter

unelegant + speicher-effizient

Aktuelle Dimension

Vereinbarte Dimension

```
call up(n,m,A(i1,j1),100)
```

```
subroutine up(n,m,A,LDA)
dimension A(LDA,*)
```

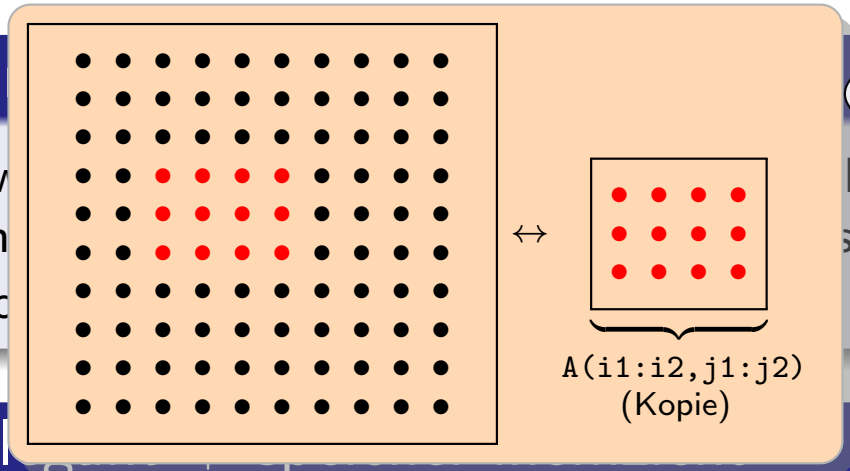
Startadresse der Submatrix

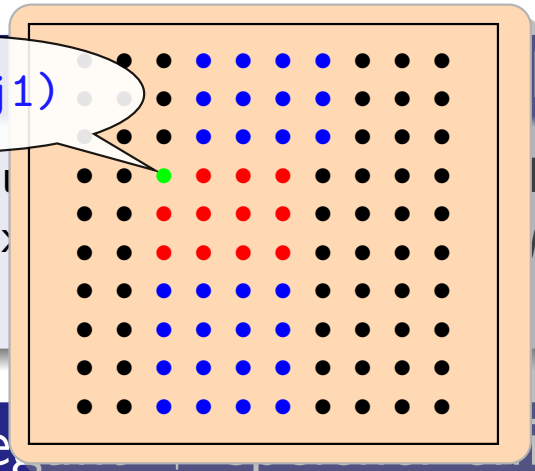
Leading Dimension of A

Originalspeicherplatz, größere Lücken zwischen Spalten (jeweils `LDA`$-n$)

Verarbeitung in beiden UP-Varianten jeweils: `A(i,j)` für `i=1,n` und `j=1,m`

Anweisungen in Fortran - Unterprogramme





U … **bergabe**

Zv … Parameter
im … $i_1 \ldots j_2$,
wo …

… lfeldern (Su …
… s Submatrix …

el … **unele** … **izient**

```
call up(n,m,A(i1:i2,j1:j2))
```

```
subroutine up(n,m,A)
dimension A(n,m)
```

temporäre, dicht gepeicherte Kopie der Submatrix als akt. Parameter

```
call up(n,m,A(i1,j1),100)
```

```
subroutine up(n …
dimension A(LDA, …
```

Startadresse der Submatrix

Originalspeicherplatz, größere Lücken zwischen Spalten (jeweils `LDA`$-n$)

Verarbeitung in beiden UP-Varianten jeweils: `A(i,j)` für `i=1,n` und `j=1,m`

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle INTERFACE

- Problematik falscher Datentypen bei aktuellen Parametern tritt auch auf, wenn Konstante in der Rufzeile übergeben werden, denn

$$1 \neq 1. \neq 1d0$$

- Gewisse Absicherung in F90 mittels `INTERFACE`-Anweisung möglich: (ähnelt der Prototyp-Vereinbarung in C)

```
PROGRAM / SUBROUTINE / FUNCTION ...
  INTERFACE
    SUBROUTINE name (a,b,c)          ∀ benutzten externen UP
    REAL :: a,b,c
    END SUBROUTINE name
  END INTERFACE
  ... eigentliche Anweisungen, z.B. call name(x,y,z)
END
```

- Wenn Interface in mehreren UP auftritt, ist `INCLUDE` zu empfehlen.

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle INTERFACE

- Problematik falscher Datentypen bei aktuellen Parametern tritt auch auf, wenn Konstante in der Rufzeile übergeben werden, denn

$$1 \neq 1. \neq 1d0$$

- Gewisse Absicherung in F90 mittels `INTERFACE`-Anweisung möglich: (ähnelt der Prototyp-Vereinbarung in C)

```
PROGRAM / SUBROUTINE / FUNCTION ...
  INTERFACE
    FUNCTION fn (b)                    ∀ benutzten externen UP
    REAL :: fn,b
    END FUNCTION fn
  END INTERFACE
  ... eigentliche Anweisungen, z.B. y=fn(x)
END
```

- Wenn Interface in mehreren UP auftritt, ist `INCLUDE` zu empfehlen.

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — INTERFACE

- Insbesondere im Zusammenhang mit MODULE-Vereinbarungen werden mittels INTERFACE auch Operatoren oder Zuweisungsoperationen „umdefiniert“, z. B. für selbst definierte Datentypen (TYPE-Anweisung)
-
```
INTERFACE OPERATOR (*)
  FUNCTION intersect (a,b)
  USE set_module   (extern definierter Module)
  TYPE(set), INTENT(in) :: a,b
  TYPE(set) :: intersect
  END FUNCTION intersect
END INTERFACE
INTERFACE ASSIGNMENT (=)
  SUBROUTINE int_to_bit (n,b)
  INTEGER, INTENT(in) :: n
  LOGICAL, INTENT(out) :: b
  END SUBROUTINE int_to_bit
END INTERFACE
```

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle — Generische Unterprogramme

Im Quelltext erscheint ein einheitlicher Rufname für verschiedene Unterprogramme (wie bei Standardfunktionen), Auswahl abhängig vom Typ der Argumente

```
PROGRAM ...
INTERFACE name
  SUBROUTINE r_name(x,y)
  REAL :: x,y
  END
  SUBROUTINE i_name(x,y)
  INTEGER :: x,y
  END
END INTERFACE name
INTEGER i,j
REAL a,b
...
CALL name(i,j)      → Compiler verwendet i_name
CALL name(a,b)      → Compiler verwendet r_name
...
END PROGRAM
```

Unterprogramm-Schnittstelle — ENTRY-Anweisung

- Ein Unterprogramm kann Alias-Namen erhalten, ggf. auch mit modifizierter Parameterliste (etwa als Gegenteil generischer UP)
- Die ENTRY-Anweisung (mit ähnlicher Syntax wie SUBROUTINE) definiert dazu einen „sekundären Eintrittspunkt“ in das Unterprogramm.
- ENTRY kann zwischen ausführbaren Anweisungen stehen (auch mehrfach):

```
SUBROUTINE name1(a,b,c)
      Vereinbarungen (incl. a,b,c,x)
      Anweisungen
ENTRY name2(a,b,c,x)      (wird bei Aufruf von name1 ignoriert)
      Anweisungen
RETURN
END
```

Anweisungen in Fortran - Unterprogramme

Unterprogramm-Schnittstelle

- Ein Unterprogramm kann Alias-Namen enthalten, ggf. auch mit modifizierter Parameterliste (etwa als Gegenteil generischer UP)
- Die ENTRY-Anweisung (mit ähnlicher Syntax wie SUBROUTINE) definiert dazu einen „sekundären Eintrittspunkt" des Unterprogramm.
- ENTRY kann zwischen ausführbaren Anweisungen stehen (auch mehrfach):

```
SUBROUTINE name1(a,b,c)
      Vereinbarungen (incl. a,b,c,x)
      Anweisungen
ENTRY name2(a,b,c,x)
      Anweisungen
RETURN
END
```

ENTRY-Anweisung

```
subroutine plotx (n,x,ier)
integer n,ier,jx(*)
real x(*)
logical PX
PX = .TRUE.
GOTO 1
entry plotj (n,jx,ier)
PX = .FALSE.
1 CONTINUE
 ...
if (PX) then
 write(*,'(F7.2)') x(i)
else
 write(*,'(I5)') jx(i)
endif
 ...
```

(wird durch Aufruf von name1 reaktiviert)

Unterprogramme mit optionalen Parametern

- Bei Vereinbarung formaler Parameter eines Unterprogramms kann man das zusätzliche Attribut `OPTIONAL` angeben:

```
SUBROUTINE up (a,b,c)
REAL, INTENT(IN), OPTIONAL :: a,c
REAL, INTENT(IN OUT) :: b
```

- Um ein Unterprogramm mit optionalen Parametern aufrufen zu können, muss das rufende Programm die zugehörige `INTERFACE`-Vereinbarung enthalten.
- Wenn die Zuordnung nicht mehr durch die Reihenfolge der aktuellen Parameter eindeutig ist, müssen beim Aufruf Schlüsselworte (Namen der formalen Parameter laut `INTERFACE`) angegeben werden, z. B. `CALL up(x,y)` (wenn c fehlt), aber `CALL up(b=y,c=z)` oder `CALL up(b=y)`, wenn (auch) a fehlt.
- Nicht angegebene optionale Parameter dürfen im Programm nicht benutzt werden, außer um mit der Standardfunktion `PRESENT` festzustellen, ob der Parameter angegeben war,

  `IF ( PRESENT(c) ) THEN ...` (Benutzung von c möglich)

  oder als optionaler aktueller Parameter eines zu rufenden Unterprogramms.

Anweisungen in Fortran - Unterprogramme

Anweisungsfunktionen

- Zwischen Vereinbarungsteil und Anweisungsteil können Anweisungsfunktionen definiert werden:

  ```
  f(argumentliste) = ausdruck
  ```

  z. B. :

  ```
  dist(x1,y1,x2,y2)=sqrt((x1-x2)**2+(y1-y2)**2)
  ```

- Aufruf erfolgt im Anweisungsteil (derselben Programmeinheit) wie üblich, z. B.

  ```
  d=dist(x(i),y(i),x(k),y(k))
  ```

- Kein Unterprogrammaufruf, sondern anstelle des Funktionsaufrufs wird im Quelltext der zuvor definierte Ausdruck mit aktuellen Parametern eingesetzt (vgl. Makroanweisung)
- Der Ausdruck in der Anweisungsfunktionsdefinition kann auch andere Variable des Unterprogramms verwenden – das ist aber „unsauber“ (nicht erkennbare Funktionsschnittstelle)

Interne Unterprogramme (CONTAINS) nur F90

Definition und Gültigkeit nur innerhalb derselben Programmeinheit, z. B. :

```
SUBROUTINE outside(a)
REAL, INTENT(IN) :: a
...
CALL inside(i)    !  kann nicht von außen gerufen werden
x = sin(3.89)     !  ruft nicht die Standardfunktion
...
CONTAINS
  subroutine inside(k)
  integer, intent(in) :: k
  ...
  end subroutine inside

  real function sin(m)
  real, intent(in) :: m
  ...
  end function sin
END SUBROUTINE outside
```

Unterprogramme / Funktionen in der Rufzeile

Wird der Name einer Funktion oder eines Unterprogramms als aktueller Parameter an ein Unterprogramm übergeben, muss dieser im rufenden Programm mit `EXTERNAL` vereinbart werden (bei Standardfunktionen mit `INTRINSIC`), um Missverständnisse zu vermeiden.

Beispiel: Eine Funktion `INTEGRAL` soll eine Näherung für $\int_a^b f(x)\,dx$ berechnen. Um welche Funktion `f` es sich handelt, muss als Parameter übergeben werden, außerdem die Intervallgrenzen und eine Schrittweite oder Anzahl von Stützstellen im Intervall:

```
REAL FUNCTION Integral(f,a,b,n)
REAL f,a,b,h,sum,x
INTEGER n,i
h = (b-a)/n
...    Berechnung der Näherung mit wiederholtem Aufruf f(a+i*h)
Integral = sum
RETURN
END
```

Unterprogramme / Funktionen in der Rufzeile

Wird der Name einer Funktion oder eines Unterprog
Parameter an ein Unterprogramm übergeben, muss
mit `EXTERNAL` vereinbart werden (bei Standardfunkt
Missverständnisse zu vermeiden.

```
REAL FUNCTION f1(x)
REAL x
f1 = sin(x)+cos(2*x)
RETURN
END
```

Beispiel: Eine Funktion `INTEGRAL` soll eine Näherung für $\int_a^b f(x)\,dx$ berechnen. Um
welche Funktion `f` es sich handelt, muss als Parameter ü
Intervallgrenzen und eine Schrittweite oder Anzahl von S

```
REAL FUNCTION f2(x)
REAL x
f2 = sqrt(1.+cos(x)**2)
RETURN
END
```

```
REAL FUNCTION Integral(f,a,b,n)
REAL f,a,b,h,sum,x
INTEGER n,i
h = (b-a)/n
...    Berechnung der Näherung mit wiederholtem Aufruf f(a+i*h)
Integral = sum
RETURN
END
```

Anweisungen in Fortran - Unterprogramme

Unterprogramme als Funktionen in der Rufzeile

Wird der Name einer Funktion oder eines Unterprogramms als Parameter an ein Unterprogramm übergeben, muss dieser im rufenden Programm mit EXTERNAL vereinbart werden (bei Standardfunktionen mit INTRINSIC), um Missverständnisse zu vermeiden.

Beispiel: Eine Funktion f (es sich handelt, muss als Parameter übergeben werden, außerdem die Intervallgrenzen und eine Schrittweite oder Anzahl von Stützstellen im Intervall:

Wir können eine Näherung für $\int_a^b f(x)\,dx$ berechnen. Um

```
      PROGRAM MAIN
      REAL a,b,Integral,f1,f2
      EXTERNAL f1,f2
      INTRINSIC cos
      INTEGER n,...
      ...   ← a,b,n bestimmen
      WRITE(*,*) n,
     +           Integral(f1,a,b,n),
     +           Integral(f2,a,b,n),
     +           Integral(cos,a,b,n)
      ...
      END
```

```
REAL FUNCTION f1(x)
REAL x
f1 = sin(x)+cos(2*x)
RETURN
END
```

```
REAL FUNCTION f2(x)
REAL x
f2 = sqrt(1.+cos(x)**2)
RETURN
END
```

```
REAL FUNCTION Integral(f,a,b,n)
REAL f,a,b,h,sum,x
INTEGER n,i
h = (b-a)/n
...   Berechnung der Näherung mit wiederholtem Aufruf f(a+i*h)
Integral = sum
RETURN
END
```

Anweisungen in Fortran - Unterprogramme

Unterprogramme als Funktionen in der Rufzeile

Wird der Name einer Funktion oder eines Unterprogramms als Parameter an ein Unterprogramm übergeben, muss dieser im rufenden Programm mit EXTERNAL vereinbart werden (bei Standardfunktionen mit INTRINSIC), um Missverständnisse zu vermeiden.

Beispiel: Eine Funktion f (es sich handelt, muss als Parameter übergeben werden, außerdem die Intervallgrenzen und eine Schrittweite oder Anzahl von Stützstellen im Intervall:

$$\int_a f(x)\,dx$$ berechnen. Um

```
      PROGRAM MAIN
      REAL a,b,Integral,f1,f2
      EXTERNAL f1,f2
      INTRINSIC cos
      INTEGER n,...
      ...   ← a,b,n bestimmen
      WRITE(*,*) n,
     +           Integral(f1,a,b,n),
     +           Integral(f2,a,b,n),
     +           Integral(cos,a,b,n)
      ...
      END
```

```
REAL FUNCTION f1(x)
REAL x
f1 = sin(x)+cos(2*x)
RETURN
END
```

```
REAL FUNCTION f2(x)
REAL x
f2 = sqrt(1.+cos(x)**2)
RETURN
END
```

```
REAL FUNCTION Integral(f,a,b,n)
REAL f,a,b,h,sum,x
INTEGER n,i
h = (b-a)/n
...   Berechnung der Näherung mit wiederholtem Aufruf f(a+i*h)
Integral = sum
RETURN
END
```

Spezielle Schnittstellen

Kommandozeilenargumente

- Fortran-Hauptprogramm hat keine Argumente: `PROGRAM program_name`
- zum Vergleich in C: `int main(int argc, char** argv) { ... }`
- Kommandozeilenargumente sind Zeichenketten, die als Array `argv` bereitgestellt werden.
- Zugriff in Fortran über Unterprogrammaufrufe (`IARGC`/`GETARG`) möglich:
- Aufruf eines Programms „`Hallo`“: `./Hallo Herr Kaiser`

C-Quelltext

```
argc ← 3
argv[0] ← "./Hallo"
argv[1] ← "Herr"
argv[2] ← "Kaiser"
```

Fortran-Quelltext

```
character*20 str
narg=IARGC() ← 2
call GETARG(1,str) str ← "Herr"
call GETARG(2,str) str ← "Kaiser"
call GETARG(0,str) str ← "./Hallo"
```

Spezielle Schnittstellen

Kommandozeilenbefehle ausführen

Das von der Sprache C her bekannte Unterprogramm `system` kann i.a. auch von Fortran-Programmen aufgerufen werden:

```
call system(kommandozeilenbefehl)
```

wobei der Befehl als C-String anzugeben ist, d. h. mit abschließendem Nullbyte,

auch als Funktionsaufruf möglich: `ier=system(kommandozeilenbefehl)`,
dann zeigt `ier=0` die fehlerfreie Kommandoausführung an.

Beispiel:
aus dem Fortranprogramm die Namen bestimmter Datenfiles aus dem aktuellen Verzeichnis anzeigen (geordnet nach letzter Änderungszeit der Dateien)

```
call system ( "ls -ltr *.dat"//char(0) )
```

Spezielle Schnittstellen

Fortran $\leftrightarrow$ C

Fortran- und C-Unterprogramme können sich gegenseitig aufrufen, wenn bestimmte Rahmenbedingungen eingehalten werden:

- Interne Namen (Variablen in Unterprogrammen) spielen keine Rolle
- Externe Namen (Unterprogrammnamen, Common-Block-Namen) müssen in C an die Gepflogenheiten des Fortran-Compilers angepasst werden (s.u.)
- Parameter von Unterprogrammen immer by-reference übergeben, passende Typvereinbarungen

Konventionen für Unterprogrammnamen

In C case-sensitiv, in Fortran nicht. Fortran-Compiler bildet die unterschiedliche Schreibweise auf einen gewissen Standard ab (leider nicht einheitlich), z.B.

Quelltext	GNU f77	Sun, gfortran	HP	NCUBE, Cray T3E
`NamE`	`name_`	`name_`	`name`	`NAME`
`A_new`	`a_new__`	`a_new_`	`a_new`	`A_NEW`

Spezielle Schnittstellen

Fortran $\leftrightarrow$ C

Fortran- und C-Unterprogramme können sich gegenseitig aufrufen, wenn bestimmte Rahmenbedingungen eingehalten werden:

- Interne Namen (Variablen in Unterprogrammen) spielen keine Rolle
- Externe Namen (Unterprogrammnamen, Common-Block-Namen) müssen in C an die Gepflogenheiten des Fortran-Compilers angepasst werden (s.u.)
- Parameter von Unterprogrammen immer by-reference übergeben, passende Typvereinbarungen

Konventionen für Unterprogrammnamen

In C case-sensitiv, in Fortran nicht. Fortran-Compiler bildet die unterschiedliche Schreibweise auf einen gewissen Standard ab (leider nicht einheitlich), z.B.

Quelltext	GNU f77	Sun, gfortran	HP	NCUBE, Cray T3E
`NamE`	`name_`	`name_`	`name`	`NAME`
`A_new`	`a_new__`	`a_new_`	`a_new`	`A_NEW`

Spezielle Schnittstellen

Fortran $\leftrightarrow$ C

Fortran- und C-Unterprogramme können sich gegenseitig aufrufen, wenn bestimmte Rahmenbedingungen eingehalten werden:

- Interne Namen (Variablen in Unterprog
- Externe Namen (Unterprogrammnamen
  C an die Gepflogenheiten des Fortran-C
- Parameter von Unterprogrammen imme
  Typvereinbarungen

Für C-Quellen Anpassung an jeweilige Fortran-Version per Header-File:

```
#ifdef UNDER
 #define name name_
 #ifdef DBL
 #define a_new a_new__
 #else
 #define a_new a_new__
 #endif
#endif
```

und Übersetzung mittels

```
gcc -D UNDER [-D DBL] ...
```

Im C-Quellcode nur `name` bzw. `a_name`

Konventionen für Unterprogrammnamen

In C case-sensitiv, in Fortran nicht. Fortran-Con
Schreibweise auf einen gewissen Standard ab (le

Quelltext	GNU f77	Sun, gfortran	HP	Cray T3E
`NamE`	`name_`	`name_`	`name`	`NAME`
`A_new`	`a_new__`	`a_new_`	`a_new`	`A_NEW`

Spezielle Schnittstellen

Fortran $\leftrightarrow$ C

Fortran- und C-Unterprogramme können sich gegenseitig aufrufen, wenn bestimmte Rahmenbedingungen eingehalten werden:

- Interne Namen (Variablen in Unterprogrammen) spielen keine Rolle
- Externe Namen (Unterprogrammnamen, Common-Block-Namen) müssen in C an die Gepflogenheiten des Fortran-Compilers angepasst werden (s.u.)
- Parameter von Unterprogra... sende Typvereinbarungen

Alle Parameter by reference

Bsp. C $\rightarrow$ Fortran (`SUBROUTINE UPF(N)`)

```
#define upf upf_
 int n=100;
...
 upf( &n );
```

Bsp. Fortran (`CALL UPC(N)`) $\rightarrow$ C

```
#define upc upc_
 void upc( int* n ) {
...
 *n = 100;
```

Konventionen für Unterprogram...

In C case-sensitiv, in Fortran nicht. ... Schreibweise auf einen gewissen St...

Quelltext	GNU f77	Sun, gfortra...
`NamE`	`name_`	`name_`
`A_new`	`a_new__`	`a_new_`

Spezielle Schnittstellen

Fortran ↔ C – Zugriff auf Daten eines COMMON-Blocks

- Für den Namen des Common-Blocks gelten die gleichen Konventionen wie für Namen von Unterprogrammen.
- Ein Fortran-Common-Block ist in C als globale (externe) `struct`-Variable verwendbar.
- Fortran:
```
COMMON /Daten/ N,X,Y
```
C:
```
#define daten daten_
extern struct { int n; float x, y; } daten;
```
- Zur besseren Lesbarkeit kann man für die `struct`-Komponenten Namen definieren, die denen in Fortran gleichen:
```
#define N daten.n
#define X daten.x
#define Y daten.y
```