HAUPT-UND UNTERPROGRAMME

- **Das Hauptprogramm (main program)**.

- **Subroutine-Unterprogramme**, intern oder extern. Man erkennt Unterprogramme an der Überschrift, z. B.

  ```
  SUBROUTINE FLAECHE(r,A).
  ```

  Eine Verzweigung erfolgt durch das CALL:

  ```
  CALL FLAECHE(r,A).
  ```

  Subroutinen sind **intern**, wenn sie hinter einem `CONTAINS`

  ```
  PROGRAM HAUPT
    ...
  CONTAINS
      SUBROUTINE ...
      ...
      END SUBROUTINE ...

  END PROGRAM HAUPT
  ```

  in ein Haupt- oder Unterprogramm eingebettet sind.

- **Funktions-Unterprogramme**: Wie bei der Subroutine gibt es interne (seit Fortran 90) und externe Formen (der Klassiker).

  ```
  REAL FUNCTION extremum(A,B).
  ```

  Der Aufruf wird einem Rechenausdruck benannt.

  ```
  Y = 4 * extremum(V1,V2) + EXP(V1).
  ```

- **Moduln**. Neuerung ab Fortran 90.

```
MODULE Speicher
  ...
END MODULE Speicher
```

  Moduln enthalten Spezifikationen, Definitionen abgeleiteter Typen oder Anweisungen zur Speicherverwaltung. Weiterhin können auch Prozedur-Bibliotheken mit zugehörigen Schnittstellen-Informationen enthalten sein. Durch die Anweisung

```
USE Speicher
```

  erhält man einen Zugriff auf die Definitionen des Moduls `Speicher`

- **Modul-Unterprogramme**, deren Code hinter dem `CONTAINS`

```
  ...
CONTAINS
    SUBROUTINE ...
    ...
END MODULE Speicher
```

  in einem MODULE enthalten ist, werden durch Angabe einer passenden USE-Anweisung in dem Programm oder Unterprogramm integriert.

- **Block Data-Programmeinheit**. Anfangsbereiche von benannten Commonbereichen zu definieren. `BLOCK DATA`, sie werden nicht aufgerufen. Sie bewirken allein durch ihre Existenz die Setzung der Anfangswerte der benannten Commonbereiche.(**Veraltet!**)

Der Klassiker: Externes Unterprogramm

```
PROGRAM Geometrie

Pi=4.*ATAN(1.)
radius=3.

CALL Kreis(radius,Umfang,Flaeche)

PRINT '(1X,2(2X,A,F10.2))',"Radius=",radius,"Flaeche=",Flaeche
END PROGRAM Geometrie

SUBROUTINE Kreis(r,U,F)    ! extern, kann separat uebersetzt werden
REAL, INTENT(in) :: r      ! freiwillig
REAL, INTENT(out) :: U, F  ! freiwillig
Pi=4.*ATAN(1.)             ! Pi aus dem Hauptprogramm ist unbekannt
U=2.*r*Pi
F=r**2*Pi
RETURN                     ! return ist hier eigentlich ueberfluessig
END SUBROUTINE Kreis       ! extern
```

Das interne Unterprogramm gibt es seit Fortran 90

```
PROGRAM Geometrie

Pi=4.*ATAN(1.)
radius=3.

CALL Kreis(radius)

PRINT '(1X,2(2X,A,F10.2))',"Radius=",radius,"Flaeche=",Flaeche

CONTAINS

   SUBROUTINE Kreis(r)           ! intern, sieht alle Variablen
   REAL, INTENT(in) :: r         ! des Hauptprogrammes ausser r
   Umfang=2.*r*Pi                ! Pi ist bekannt
   Flaeche=r**2*Pi
   RETURN
   END SUBROUTINE Kreis

END PROGRAM Geometrie
```

Globale Variablen zur Kommunikation im Module

```
!----------START-------------
MODULE comm
  REAL, PARAMETER :: Pi=3.14159
  REAL :: radius, Umfang, Flaeche
END MODULE comm

PROGRAM Geometrie  ! da beginnt das Hauptprogramm
USE comm           ! MODULE zur Kommunikation

radius=3.; CALL Kreis

PRINT '(1X,2(2X,A,F10.2))',"Radius=",radius,"Flaeche=",Flaeche
END PROGRAM Geometrie

SUBROUTINE Kreis
USE comm, r=>radius, U=>Umfang    ! radius in r umbenannt, ..
U=2.*r*Pi; Flaeche=r**2*Pi
END SUBROUTINE Kreis
```

Ein Modulunterprogramm hat eine explizite Schnittstelle

```
!--------START--------
MODULE comm

REAL, PARAMETER ::  Pi=3.14159

CONTAINS

   SUBROUTINE Kreis(r,A,U)    ! MODULE-Procedure
   U=2.*r*Pi
   A=r**2*Pi
   RETURN
   END SUBROUTINE Kreis

END MODULE comm


PROGRAM Geometrie   ! Hier beginnt das Hauptprogramm
USE comm

radius=3.; CALL Kreis(U=Umfang,r=radius,A=Flaeche)

PRINT '(1X,2(2X,A,F10.2))',"Radius=",radius,"Flaeche=",Flaeche
PRINT '(3X,A,F10.5)',"Pi=",Pi
END PROGRAM Geometrie
```

- **Standard-Prozeduren (intrinsic functions, intrinsic subroutines)**. Zu diesen Unterprogrammen zählen z. B. `SIN`, `COS`, `SQRT`, die wie Funktions-Unterprogramme aufgerufen werden. Daneben existieren einige Subroutineformen, die mit CALL aufgerufen werden, z. H. `CALL RANDOM_NUMBER(...)`.

- **Anweisungsfunktionen (Formel-Funktionen, "Statement function")**. Umfaßt nur eine Zeile und ist somit nur bedingt als Programmeinheit zu bezeichnen.(**Veraltet!**)

- **INCLUDE-Code**. Durch INCLUDE-Anweisung können Code-Zeilen in Programme eingefügt werden (Editor-Funktionalität) Beispiel: `INCLUDE "CODE.SEQ"`.

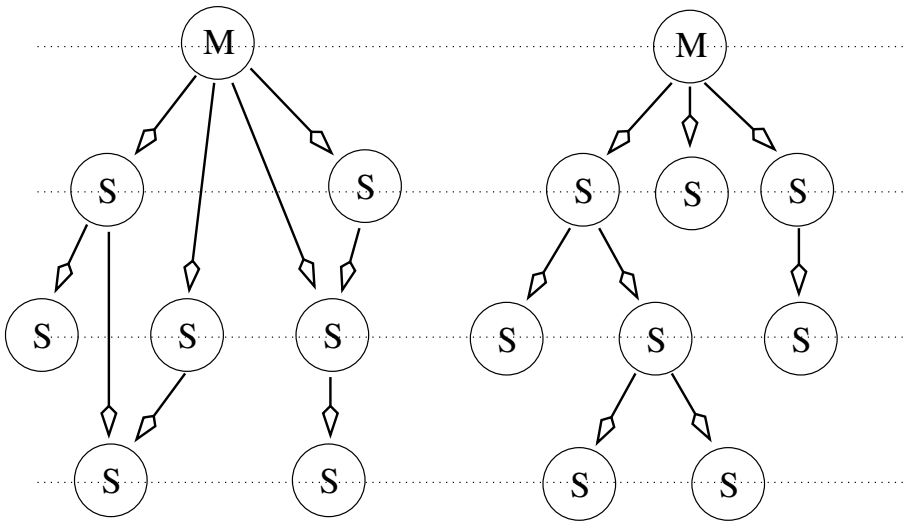
Die externen Subroutine- und Funktions-Unterprogramme werden unter dem Begriff externe Prozeduren zusammengefasst. Deren Quelle braucht nicht als Fortran Quelle vorhanden sein. Anschluss z.B als Bibliothek

```
f95 meinprogram.f90 expertlibrary.a
```

Unterprogramm-Konzept

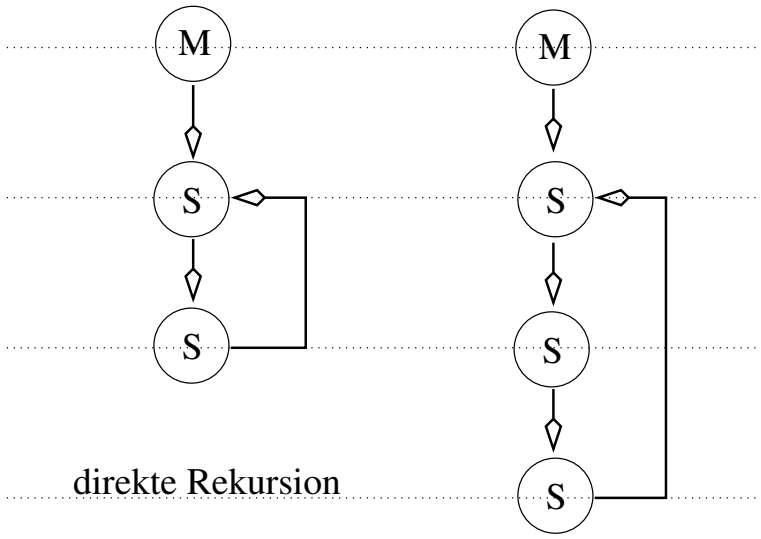
- Aufruf als 'Black Box'
- identische Programmteile wiederholen
- lokale Datenobjekte zeitlich begrenzen
- rekursive Programmteile (seit Fortran90)

Aufrufhierarchien



Hierarchie baumartige strenge Hierarchie

Rekursion



direkte Rekursion

indirekte Rekursion

Programm-Einheiten

```
PROGRAM, SUBROUTINE, FUNCTION, MODULE
END PROGRAM,...END MODULE
```

- Interne Unterprogramme und Modul-Unterprogramme sind keine Porgrammeinheiten: Sie können nur **mit** der sie umgebenden Programmeinheit* zusammen übersetzt werden.

- Dafür (weil der Compiler das Unterprogramm und das rufende Programm kennt) ist die **"Schnittstelle explizit"**, es stehen automatisch erweiterte Aufrufformen zur Verfügung: Aufruf mit Schlüsselworten, optionale Parameter.

- Interne Unterprogramme dürfen keine weiteren internen Unterprogramme enthalten, sie dürfen aber andere interne Unterprogramme der umgebenden Programmeinheit aufrufen.

- Externe Subroutine und Function Unterprogramme sind eigene Programmeinheiten. Sie können separart (sowie mit anderen Optimierungseinstellungen) übersetzt werden. Falls es "viele" sind lohnt sich das Ablegen als "Programmbibliothek". Der Aufrufer muss (copyright: darf) lediglich eine Beschreibung der Liste der "Ubergabeparameter" vorliegen haben. Durch einen **Interface-Block** im rufenden Programm, kann die Schnittstelle **explizit gemacht werden**. Dann können auch optionale Parameter und Schlüsselwortaufruf genutzt werden.

*Beim Modul-Unterprogramm ist dies nur das umgebende Modul

Ratschläge zu (externen) Unterprogrammen

- Einfache Schnittstellen
- unabhängig voneinander
- information hiding (Black Box)
- nachträgliche Änderungen möglich
- maximal ca. 200-300 Anweisungen

Übersicht: Einbettung der Unterprogramme

- **Externe Unterprogramme**

  nach (oder vor) Hauptprogramm oder separater Datei
  Beispiel: LRZ-Graphik, IMSL, NAG

- **Interne Unterprogramme**

  eingebettet in Haupt- oder Unterprogramm nach `CONTAINS`;
  keine weiteren internen Unter-Unterprogramme erlaubt!

- **Modul-Unterprogramme***
  eingebettet in Modul nach `CONTAINS`, dürfen interne Unterprogramme enthalten(!).

- **Vordefinierte Unterprogramme**
  Teil von Fortran (SIN, LOG)

*Das MODUL das die Modul-Unterprogramme (Module Procedures) enthält, darf separat gespeichert und übersetzt werden.

Formalparameter (dummy arguments)

Beim Aufruf heißen diese: **Aktualparameter**. Auch der Name "Übergabeparameter" ist üblich.

Man unterscheidet:

- Eingansparameter
- Ausgangsparameter
- Transiente Parameter (Eingabewert wird geändert!)

Fortran90 **erlaubt** die Unterscheidung mittels der Attribute:

- `INTENT(IN)`
- `INTENT(OUT)`
- `INTENT(INOUT)`

Formen von "Dummy Arguments"

- **Einfache Form:**

  Anzahl, Reihenfolge, Typ von Dummy-Parametern und Aktualparametern müssen übereinstimmmen, der Name nicht!
  Zusatz: Skalar/Feld-Eigenschaft muss übereinstimmen.

- **Fortgeschritten:**

  (seit Fortran90): Schlüsselwort Parameter (Keyword argument) und optionale Parameter

  Voraussetzung: **explizite Schnittstelle**, d.h. entweder

  - Modul-Unterprogramm
  - internes Unterprogramm
  - externes Unterprogramm + Interface Block

  optionale Parameter sind gekennzeichnet durch das Attribut:

  - `OPTIONAL`

Sichtbarkeitsbereich (=Scope) von Datenobjekten

```
PROGRAM HAUPT   ! host scoping unit
                ! von intern_zu_Haupt
                ! Haupt ist ein globaler Name
REAL :: x
INTEGER :: i

...
CONTAINS

SUBROUTINE intern_zu_Haupt  ! scoping unit
INTEGER :: i    ! i nicht mehr global
INTEGER :: j    ! nur in intern_zu_Haupt
PRINT*,x        ! Wert wie im Hauptprogramm
..
END SUBROUTINE intern_zu_Haupt ! noetig

SUBROUTINE auch_intern_zu_Haupt
PRINT*,i        ! Wert wie in Haupt
PRINT*,x        ! Wert wie in Haupt
PRINT*,j        ! Fehler: unbekanntes j
END SUBROUTINE auch_intern_zu_Haupt

END PROGRAM HAUPT

SUBROUTINE extern  ! extern ist ein globaler Name
! hier sind x und i unbekannt
..
END SUBROUTINE extern
```

Hausaufgabe

Wird ein Luftpaket der (absoluten) Temperatur T_K adiabatisch gehoben bis das enthaltene Wasser kondensiert, so hat es die Temperatur T_L, die etwas niedriger ist als T_K (weil weiter oben). Für T_L gilt näherungsweise die Formel

$$T_L = \frac{1}{\frac{1}{T_K - 55} - \frac{\ln(U/100)}{2840}} + 55.$$

Hierin ist U die relative Feuchte (in Prozent). Schreiben Sie ein Unterprogram mit Eingabeparameter U das T_L ausgibt. Dies soll vom Hauptprogramm gerufen werden, um eine Tabelle der Form

```
Werte von TL fuer relative Feuchten 10-90 Prozent
  TK      10     20     30      40      ......... 90
-------------------------------------------------
 200     XXX.X  XXX.X  ....
 210     XXX.X
 ...
 310
```

zu erstellen.

Hausaufgabe

Die Luftfeuchtigkeit wird häufig mit einem Psychrometer bestimmt. Dies besteht im Prinzip aus 2 Thermometern, wovon eines befeuchtet (t_f) oder vereist (t_i, "ice" bei Frost) ist. Aus der Temperaturdifferenz in K bekommt man den Dampfdruck e in hPa.

$$e = E_W - C_f p(t - t_f)$$

ohne Eis ($t_f > 0$°C), sonst

$$e = E_i - C_i p(t - t_i)$$

Es soll ein externes Unterprogramm geschrieben werden, das als Eingabeparameter t und t_f bzw. t_i und den Luftdruck p in hPa hat, und den Dampfdruck e, die absolute Feuchtigkeit

$$\rho_W = \frac{e}{R_W T_K},$$

die relative Luftfeuchtigkeit (in Prozent)

$$f = \frac{e}{E_W} 100,$$

den Taupunkt (in °C)

$$t_d = \frac{237.2°\text{C} \log_{10} \frac{e}{6.1\text{hPa}}}{7.5 - \log_{10} \frac{e}{6.1\text{hPa}}}.$$

Der Sättigungsdampfdruck über Wasser* und Eis:

$$E_W = 6.1\text{hPa}\ 10^{\frac{7.5t}{t+237.2°\text{C}}}$$

$$E_i = 6.1\text{hPa}\ 10^{\frac{9.5t}{t+265.5°\text{C}}}.$$

Die Absolute Temperatur ist $T_K = t + 273.15$. Die Dichte von Luft ist

$$\rho = \frac{p}{RT}$$

Das Mischungsverhältnis ist $q = \rho_W/\rho$, die spezifische Feuchte $s = \frac{\rho_W}{\rho+\rho_W}$. Die Psychrometerkonstanten sind $C_f = 0.662 \cdot 10^{-3} K^{-1}$, $C_i = 0.570 \cdot 10^{-3} K^{-1}$. Die Gaskonstanten für (trockene) Luft ist $R = 286$ J kg^{-1} K^{-1} und für Wasserdampf $R_W = 461.6$ J kg^{-1} K^{-1}.

Testfall: $t = 20$°C, $t_f = 15$°C, $p = 950$ hPa.

Freiwilliger Zusatz: Dasselbe als Modul-Unterprogramm, alle Ausgaben optional.

*Formeln nicht mehr ganz aktuell

Übergabe von Feldern

```
REAL, DIMENSION :: A(100), B(-5:5), C(1995:2002,0:12)
CHARACTER(LEN=10) :: stringvar="ok"
..
CALL SUB1(A,N)       !  = CALL SUB1(A(1),N)    ab A(1), siehe DIMENSION
CALL SUB1(B,N/2)     !  = CALL SUB1(B(-5),N/2) ab B(-5) siehe DIMENSION
CALL SUB1(A(3),5)    !                         ab A(3)
CALL SUB2(A,4,'Das ist ok so'); CALL SUB2(A,4,stringvar)
CALL SUB1(C(1997:2001,5:9),J)     ! Teil-feld

SUBROUTINE SUB1(X,L)
DIMENSION(1:L) :: X  ! uebernommenes Feld, L Elemente
DIMENSION(-L:L) :: Y ! automatisches Feld, verschwindet bei END wieder
DIMENSION(10) :: Z   ! "gefangenes" Feld, Inhalt darf bei END vergessen
DIMENSION(10), SAVE :: S  ! -"- aber Inhalt bleibt nach END erhalten
...
END SUBROUTINE SUB1

SUBROUTINE SUB2(Y,M,text)
REAL, DIMENSION(M,M,2) :: Y  ! Zugriff (jetzt) 3-dimensional
CHARACTER(LEN=*) :: text     ! so uebergibt man eine Zeichenkette
PRINT "(A,I3,A)","Der text hat die Laenge :",LEN(text)," Zeichen"
PRINT "(A,A)", "Die ersten drei Buchstaben davon:",text(1:3)
..
```

- Feldübergabe = Übergabe ab einer Startadresse.
- Seit Fortran90 können "passende" Felder automatisch im Unterprogramm erzeugt werden. Deshalb braucht man keine Arbeitsfelder (Work-Arrays) mehr übergeben (sensationell!!).
- Es können auch passende Teilfelder übergeben werden.
- Die "DIMENSION" muss nicht übereinstimmen aber der **Typ (REAL, INTEGER,..)**
- Häufigster Programmier-Fehler: Zugriff auf nicht vorhandene Feldlemente

Globale Größen (mit Nebenwirkungen)

Verdeckte vordefinierte Funktionen

```
REAL, DIMENSION(-100:100) :: sin
REAL, EXTERNAL :: COS ! nimmt den selbstgeschriebenen
...
p=SIN(1)      ! Feldelement
q=SIN(1.)     ! Syntaxfehler
...
...
FUNCTION COS(x) RESULT (WERT)  ! ersetzt eigebauten COS
   WERT=...                    ! eher unrealistisch
...
END FUNCTION COS
```

Die `EXTERNAL` Anweisung hat 2 Bedeutungen für ein selbstgeschriebenes `FUNCTION` Unrerprogramm

1. Eine vordefinierte Funktion (wie `COS`) wird durch eine eigene ersetzt.

2. Die Funktion darf als Formalparameter (Dummy-Argument) an ein Unterprogramm übergeben werden

Datenobjekte in Unterprogrammen bei wiederholtem Aufruf: **SAVE**

```
SUBROUTINE Vergessen(x,y)
REAL :: x,y
REAL :: Alzheimer
INTEGER, SAVE :: zaehler=0  ! wert wird aufbewahrt
...
zaehler=zaehler+1
IF(zaehler==1) Alzheimer=1
PRINT*,"Es wurde ",zaehler,"-mal aufgerufen"
PRINT*,"Alzheimer=",Alzheimer   ! darf vergessen werden
...
```

Rekursion

Vor-Satz: `RECURSIVE` vor SUBROUTINE/FUNCTION.

```
REAL :: fakultaet
...
PRINT*,"4! =",fakultaet(4)
..
END ! rufendes Programm


RECURSIVE FUNCTION fakultaet(n) RESULT (wert)
INTEGER,INTENT(IN) :: n
INTEGER :: wert
IF(n==1) THEN
   WERT=1
ELSE
   WERT=n*fakultaet(n-1) ! Katastrophe wenn (n+1)
ENDIF
END FUNCTION fakultaet
```

Prozedurschnittstellen

Die Gesamtheit aller Informationen, die es gestatten, die korrekte Form des Aufrufs eines Unterprogramms zu bestimmen, nennt man Schnittstelle (interface) des Unterprogramms.

man unterscheidet:

- explizite Schnittstelle: internes Unterprogramm, Modul-Unterprogramm, vordefinierte Funktion
- implizite Schnittstelle (keine Quelle da): externe Unterprogramme (auch nicht-Fortran), Unterprogramm als Formalparameter

Ein Schnittstellenblock macht eine implizite Schnittstelle explizit.

Ein Schnittstellenblock ist vorgeschrieben für

- optionale Formalparameter
- FUNCTION hat Feld oder Zeiger als Resultat
- Unterprogramm definiert einen Operator
- Unterprogramm wird mit generischem Namen gerufen
- Besondere Formalparameter: Zeiger, Zielvariable, Übernommenes Feld

Beispiel: Interface Block

```
PROGRAM Geometrie

INTERFACE circle  ! Name optional
   SUBROUTINE Kreis(r,U,F)
   REAL, INTENT(in) :: r
   REAL, INTENT(out) :: U, F
   END SUBROUTINE Kreis
END INTERFACE circle

Pi=4.*ATAN(1.)
radius=3.

CALL circle(u=Umfang,r=radius,F=Flaeche) ! MIT INTERFACE

! CALL Kreis(radius,Umfang,Flaeche)  ! OHNE INTERFACE


PRINT '(1X,2(2X,A,F10.2))',"Radius=",radius,"Flaeche=",Flaeche

END PROGRAM Geometrie

SUBROUTINE Kreis(r,U,F)  ! extern, kann separat uebersetzt werden
REAL, INTENT(in) :: r
REAL, INTENT(out) :: U, F
Pi=4.*ATAN(1.)
U=2.*r*Pi
F=r**2*Pi
RETURN
END SUBROUTINE Kreis
```

Module

Ein Modul (module) ist eine nichtausführbare Programmeinheit, die der Zusammenfassung von Datenstrukturen und Prozeduren (=Unterprogramme) zu einer Einheit dient.

```
MODULE modul_name
 Vereinbarungen, Interface-Blocks
CONTAINS
 Modul-Unterprogramme (darf interne Unterprogramme enthalten !)
END MODULE modul_name
```

Anschluss mit

```
USE modul1, lokaler_name1=>name_im_mod1
USE modul2, ONLY: lokaler_name2=>name_im_mod2
            ! nur bestimmte Objekte
USE modul_name3        ! einfach alles von modul_name3
```

Sichtbarkeitsbeschränkungen (accessability attributes)

```
MODULE beispiel1
PRIVATE :: a_inten, b_intern      ! Variablen
PRIVATE ::HilfsProcedure          ! nur im Modul bekannt
PUBLIC :: Flaeche, Pi             ! mit USE erreichbar

CONTAINS

   SUBROUTINE FLAECHE(....)
    CALL HilfsProcedure
    ...
   SUBROUTINE HilfsProcedure
   ....
END MODULE beispiel1
```

Reihenfolge der Anweisungen

<table>
<tr><td colspan="2">PROGRAM, FUNCTION, SUBROUTINE, MODULE</td></tr>
<tr><td colspan="2">USE-Anweisungen</td></tr>
<tr><td rowspan="3">FORMAT-
Anweisungen</td><td>IMPLICIT NONE</td></tr>
<tr><td>Definitionen selbstdefinierter Typen
Schnittstellenblöcke, Vereinbarungen</td></tr>
<tr><td>ausführbare Anweisungen</td></tr>
<tr><td colspan="2">CONTAINS-Anweisungen</td></tr>
<tr><td colspan="2">Interne oder Modul-Unterprogramme</td></tr>
<tr><td colspan="2">END-Anweisung</td></tr>
</table>

Erlaubte Verwendungen

	Haupt-pr.	Modul	Externes Unterpr.	Modul-Unterpr.	Internes Unterpr.	Interface -block
`USE`	J	J	J	J	J	J
`FORMAT`	J	N	J	J	J	N
Vereinb.	J	J	J	J	J	J
`INTERFACE`	J	J	J	J	J	J
ausführb.	J	N	J	J	J	N
`CONTAINS`	J	J	J	J	N	N