



Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart, Ulrich Körner, Nathanael Hübbe



Dr. Hermann-J. Lenhart

hermann.lenhart@zmaw.de



FORTRAN Einführung I:

- FORTRAN Historie
- Sprachelemente
- Darstellung und Abbildungen
- Ausdrücke
- Kontrollstrukturen
- Programmstrukturen
- „Hilfsmittel“



Kurze FORTRAN Historie (I):

FORTRAN = FORMula TRANslator

1966 als erste standardisierte Programmiersprache definiert,
 heute bekannt als FORTRAN66.

Ziel war es im Vergleich zu kryptischer Hardware-naher Programmierung (Assembler)
 quasi „in mathematischen Formeln“ zu programmieren.

Aber: $x = x + 1$ ist mathematisch nicht korrekt !!!



Kurze FORTRAN Historie (II):

1978 Neuer Standard FORTRAN77

Bringt unterschiedliche Dialekte zusammen

die Bausteine für Realisierungen von „large scale programs“ enthielten.

1990 Neuer Standard FORTRAN90

Neue Sprachentwicklung incl. Array Sprache und Abstrakte Datentypen sowie

Modernisierung des Designs aus anderen Sprachen wie Pascal, C, C++

z.B. free-form: Begrenzung von 72 auf 132 Zeichen pro Zeile heraufgesetzt



Kurze FORTRAN Historie (III):

1995 Neuer Standard FORTRAN95

<= Hier setzen wir auf !!

- Handhabung von Floating Point Ausnahmen
- Einige alte Konstrukte werden nicht mehr unterstützt (assigned goto, ...)
- Erlaubt die allocation von Arrays als strukturierte Komponente
- Verbessertes String handling



Kurze FORTRAN Historie (IV):

2004 Neuer Standard FORTRAN2003

Grundlegende Überholung von FORTRAN95,
insbesondere wird die Interoperationalität mit C standardisiert

2010 Neuer Standard FORTRAN2008 zur Verbesserung paralleler Programmierung



Elemente der FORTRAN Sprache (I)

Es gibt insgesamt 5 Datentypen in FORTRAN (intrinsic)

3 numerische Datentypen

Integer

Real

Complex

-> für numerische Berechnungen

2 nicht numerische Datentypen

Character

Logic

-> für Textbearbeitung und
für Kontrollzwecke

Zur Leistungsstärke von FORTRAN tragen selbst definierbare
sog. abgeleitete Datentypen (derived data types) bei.



Elemente der FORTRAN Sprache (II)

Integer: Ganzzahliger Wert mit oder ohne Vorzeichen

1 Wertebereich: -2147483648 bis 2147483648

0 für 32 bit Variable

-999 implizite Typvereinbarung:

3288 reservierter Name für integer Variablen i-n

+10 (Hinweis: implicit none setzen!)

integer, parameter :: k6=selected_int_kind(6)

d.h. INTEGER :: it_k6 hat den Wertebereich: -999999 bis 999999



Elemente der FORTRAN Sprache (III)

REAL: Die Default Darstellung der Floating Point Form

Integer Teil (mit/ohne Vorzeichen), Dezimalpunkt, Bruchteil (,Exponent)

1.

-0.1 Wertebereich: 10^{-37} bis 10^{+37} für IEEE Standard

1e-1 implizite Typvereinbarung:

3.1415 reservierter Name für real Variablen a-h und o-z

z.B. integer, parameter :: ehp = selected_real_kind(9, 99)

REAL :: Rmax_ehp 9 signifikante Dezimalstellen,

Wertebereich des Exponenten: 10^{-99} bis 10^{+99}

weitere Möglichkeit über Deklaration „double precision“.



Elemente der FORTRAN Sprache (IV)

COMPLEX:

Erste Konstante vor dem Komma entspricht dem Realanteil und die zweite Konstante entspricht dem Imaginär-Anteil der komplexen Größe.

(1. , 3.2)

(1 , 99e⁻²)



Elemente der FORTRAN Sprache (V)

Nicht numerische Datentypen:

Character: Anzahl von Buchstaben und Zahlen in Hochkomma.

z.B. "anything goes"

Bei character Variablen muss das erste Zeichen ein Buchstabe sein!

! ab FORTRAN2003 Case sensitiv, vorher nicht!

Logical: logischer Wert, zwei Möglichkeiten: `.true.` oder `.false.`



Elemente der FORTRAN Sprache (VI)

Derived data types:

Beispiel Person:

```
type person
```

```
    character(len=109) :: name
```

```
    real                :: age
```

```
    integer             :: id
```

```
End type person
```

```
you = person ("Smith", 23., 2534)
```

Beispiel Datum:

```
type datum
```

```
integer :: year, month, day
```

```
End type datum
```

```
z.B.: datum%year=2003
```

```
Zum Geburtstag: you%age +1.
```



FORTRAN Darstellung

Über den Typparameter (KIND-Wert) werden den Datentypen Ihre Größe im Rechner zugewiesen.

z.B. Integer (kind=4) :: i_small	~ 4 Byte Integer
Integer (kind=8) :: i_big	~ 8 Byte Integer
Real (kind=8) :: r_medium	~ 8 Byte Real
Real (kind=16) :: r_big	~ 16 Byte Real
Complex (kind=4) :: c_small	~ 2+4 byte Complex
Complex (kind=8) :: c_big	~ 2+8 Byte Complex



FORTRAN Array Abbildung

Zur Deklaration von Arrays (Feldern) müssen neben dem Datentyp auch die Größe des benötigten Feldes angegeben werden.

integer, dimension (100) :: ii, kk

real, dimension (5,4) : rr

character (len=80) :: line



FORTRAN Ausdrücke (I)

Definition Ausdruck (Expression): Beschreibung der Berechnung die vom Rechner ausgeführt werden sollen.

In FORTRAN werden Ausdrücke durch Operanden und Operatoren geformt:

$x + y$

binäre Operation

$-y$

einstelliger (monadic) Operator



FORTRAN Ausdrücke (II)

Arithmetische Ausdrücke werden durch arithmetische Operatoren gebildet,
hier die Übersicht nach Priorisierung, d.h. ohne Klammerung:

Operator	Berechnung	Priorität	Beispiel
**	Potenzierung	1	A**B
*	Multiplikation	2	A*B
/	Division	2	A/B
+	Addition	3	A+B
-	Subtraktion	3	A-B



FORTRAN Ausdrücke (III)

Vergleich arithmetische Ausdrücke vs mathematische Ausdrücke:

FORTRAN	Mathe
$a = b + c$	$a = b + c$
nicht valide	$b + c = a$
$n = 1$	$n = 1$
$n = n + 1$	nicht valide



FORTRAN Ausdrücke (IV)

Logische Ausdrücke werden durch logische Operatoren gebildet, die logische Variable oder Konstante verknüpfen und als Ergebnis einen Wert `.true.` Oder `.false.` liefern.

Operator	Ausführung	Priorität	Beispiel
<code>.NOT.</code>	Negation	1	<code>.NOT. L1</code>
<code>.AND.</code>	Konjunktion	2	<code>L1 .AND. L2</code>
<code>.OR.</code>	Disjunktion	3	<code>L1 .OR. L2</code>
<code>.EQV.</code>	Äquivalenz	4	<code>L1 .EQV. L2</code>
<code>.NEQV.</code>	Antivalenz	4	<code>L1 .NEQV. L2</code>



FORTRAN Kontrollstrukturen (I)

Kontrollstrukturen dienen dazu den Programmablauf zu steuern und in logische Einheiten (Blöcken; block construct) zu gliedern.

Dazu gehören IF Abfragen und Do Schleifen.



FORTRAN Kontrollstrukturen (II)

IF Abfragen werden genutzt um nach Abfrage der Eingangswerte einzelne Programmteile anzusteuern.

```
[Name] IF (logischer skalarer Block) THEN
    Block von Anweisungen
[ELSE [Name]
    Block von Anweisungen]
END IF [Name]
```



FORTRAN Kontrollstrukturen (III)

IF Abfrage Beispiel:

oder

```
IF (x < y) THEN
```

```
    x = -y
```

```
ELSE
```

```
    y = -y
```

```
END IF
```

```
swap: IF (x < y) THEN
```

```
    x = -y
```

```
ELSE swap
```

```
    y = -y
```

```
END IF swap
```



FORTRAN Kontrollstrukturen (IV)

IF Abfrage Notation mit ELSE IF:

```
[Name] IF (logischer skalarer Block) THEN  
    Block von Anweisungen  
[ELSE IF [Name]  
    Block von Anweisungen]  
[ELSE [Name]  
    Block von Anweisungen]  
END IF [Name]
```



FORTRAN Kontrollstrukturen (V)

IF Abfrage Beispiel mit ELSE IF:

```
IF (i < 0) THEN
```

```
    x = 0.0
```

```
ELSE IF (K < 0)
```

```
    z = 1.0
```

```
ELSE
```

```
    x = 0.0
```

```
    y = 1.0
```

```
ENDIF
```



FORTRAN Kontrollstrukturen (VI)

Vergleichsoperatoren dienen zum Vergleich von numerischen Variablen und Konstanten bzw. Zeichenausdrücke und Konstante

Operator	Kurzform	Ausführung	Beispiel
.LT.	<	kleiner als	A .LT. B
.LE.	<=	kleiner oder gleich	A .LE. B
.EQ.	==	gleich *	A .EQ. B
.NE.	/=	ungleich	A .NE. B
.GT.	>	größer als	A .GT. B
.GE.	>=	größer oder gleich	A .GE. B

* Vorsicht bei Vergleich von REAL Ausdrücken



FORTRAN Kontrollstrukturen (VII)

DO Schleife als Strukturelement in FORTRAN um Iterationen auf einfache Weise zu realisieren.

Nomenklatur der DO Schleife:

```
[Name] DO [Variable = Ausdruck1, Ausdruck2, Ausdruck3]
        Block von Anweisungen
END DO [Name]
```

Variable, Ausdruck1, Ausdruck2, Ausdruck3 sind vom Typ INTEGER



FORTRAN Kontrollstrukturen (VIII)

DO Schleife Beispiel:

```
DO i = 1,10  
    sum = sum + i  
END DO
```

i ist Laufvariable

1 ist Startwert

10 ist Endwert der Iteration

```
DO i = 1, 10,2  
    sum = sum + i  
END DO
```

der Wert 2 steuert

zusätzlich die

Schrittweite!



FORTRAN Kontrollstrukturen (IX)

```

outer_loop do i=1, imax
    inner_loop do n=1,iteration_max
        <iteration algorithm>
        ...
        if ( lconverged ) then
            print*, `iteration has converged for n at i`

            exit inner_loop                [cycle outer_loop]
        end if
    end do inner_loop

end do outer_loop

```

Alternativ do while



FORTRAN Programm-Struktur (I)

Ein **FORTRAN Programm** kann aufgebaut werden aus den Komponenten:

- Hauptprogramm
- Externe Subprogramme (Funktionen und Subroutinen)
- Module



FORTRAN Programm-Struktur (II)

Ein **FORTRAN Haupt-Programm** besteht aus:

[*program Programm – Name*]

```
program test
```

FORTRAN Ausdruck

```
print*, 'Hello World'
```

End [*Programm – Name*]

```
end program test
```

```
als Programm: test.f90
```



FORTRAN Programm-Ausführung

FORTRAN Programm Ausführung:

Compileraufruf zur Übersetzung des Quellcodes in test.f95

```
gfortran test.f95
```

liefert Executable a.out

Zur Verwendung des gleichen Namens im Executable wie das Programm:

```
gfortran -o test.x test.f95
```

```
./test.x
```

Programm ausführen



FORTRAN Programm-Hilfsmittel(I)

FORTRAN Kommentarzeilen:

```
do i=1, imax                ! Initialisierung Outer Loop
  do n=1,iteration_max      ! Initialisierung Inner Loop
    <iteration algorithm>
    ...
  ! Ende Inner Loop
  end do
! Ende Outer Loop
end do
```



FORTRAN Programm-Hilfsmittel(II)

FORTRAN Universal Debugger: Das Print Statement



```

do i=1, imax                ! Initialisierung Outer Loop
  Print*, ' Outer Loop : ',i
  do n=1,iteration_max      ! Initialisierung Inner Loop
    Print*, ' Inner Loop : ',n
    <iteration algorithm>
    ...
  ! Ende Inner Loop
  end do
! Ende Outer Loop
end do

```