Sprachelemente in FORTRAN

Zusammenstellung: Manfried Faber und Gerhard Kahl

26. Mai 2006

Inhaltsverzeichnis

1 Grundlagen **3**
1.1 Erstellen von Programmen 3
1.1.1 Ein einfaches FORTRAN-Programm 4
1.1.2 Der Aufbau eines FORTRAN-Programmes 5

2 Einfache Sprachelemente **7**
2.1 Einfache Zuweisungen und Objekte 7
2.2 Einfache Felder = Vektoren = Arrays 8
2.2.1 Syntax 8
2.2.2 Initialisieren von Arrays 10
2.2.3 Felder von Zeichen 11
2.2.4 Mehrdimensionale Arrays 11
2.3 Beenden eines Programms 13
2.4 Variablenübergabe in Fortran-Programmen 13

3 Sprachelemente in FORTRAN **16**
3.1 Datentypen und Wertebereiche in FORTRAN 16
3.1.1 FORTRAN Zeichensatz 16
3.1.2 Datentypen 16
3.1.3 Konstanten 17
3.1.4 Symbolische Konstanten 18
3.1.5 Wertebereich der Datentypen 18
3.1.6 Assoziierung 19
3.2 Ausdrücke in FORTRAN 21
3.2.1 Arithmetische Ausdrücke und Operatoren 21
3.2.2 Operationen mit Zeichenketten 22
3.2.3 Vergleichsausdrücke 23
3.2.4 Prioritäten und Verknüpfungsreihenfolge 24
3.3 Anweisungen in FORTRAN 25
3.4 Steueranweisungen 27
3.5 Nicht ausführbare Anweisungen in FORTRAN 31
3.6 Anweisungsfunktionen und interne Funktionen 33
3.7 Ein/Ausgabe in FORTRAN 37
3.7.1 Ein/Ausgabe-Befehle 37
3.7.2 Read, Write und Print 37
3.7.3 Open, Close und Inquire 40

3.7.4 Rewind, Backspace und Endfile 41
3.7.5 Format-Anweisungen 42

1 Grundlagen

1.1 Erstellen von Programmen

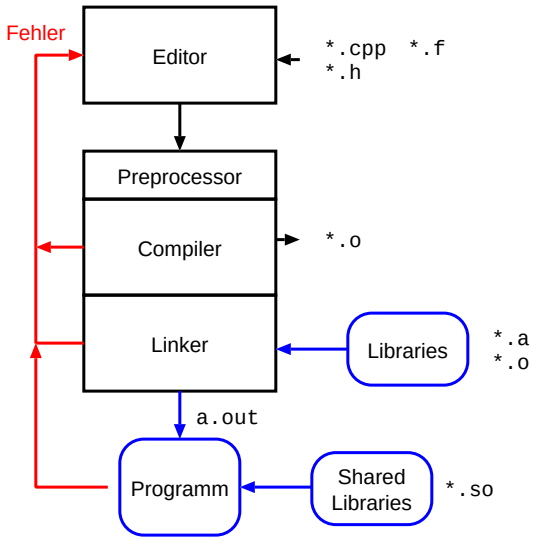

Abbildung 1.1: Compilieren & linken

Der **Editor** (z.B. `emacs, gvim, vim, vi, nedit`) erlaubt die Eingabe und die Bearbeitung der Quelltexte (`*.c, *.C, *.h; *.cpp, *.hpp oder *.f`).

Der **Compiler** (z.B. `gcc, f77`) dient zum Erzeugen eines lauffähigen Programms aus einem Quelltext. Viele moderne Produkte enthalten neben dem eigentlichen Compiler auch einen Preprocessor und den Linker.

Der **Preprocessor** bindet externe Quelltextpassagen ein, schließt Quelltextpassagen nach Maßgabe vordefinierter Bedingungen ein oder aus und ersetzt symbolische Konstanten durch wirkliche Werte. Das Ergebnis ist ein Quellprogramm, das dem eigentlichen Compiler zugeführt wird.

Der eigentliche **Compiler** bildet aus dem Quelltext den Maschinencode (Object-Code, `*.o`). Dieser enthält jedoch erst einstweilige Verweise auf externe Funktionen, deren Code im gegenständlichen Modul nicht enthalten war, z.B. Bibliotheksfunktionen oder eigenen Code in anderen Dateien.
Der **Linker** verbindet die einzelnen Object-Codes und fügt den Code der Bibliotheksfunktionen (`lib*.a`) ein. Das Ergebnis ist ein ausführbarer Code (wenn kein anderer Name angegeben wird, `a.out`), der die Informationen zum richtigen Laden und Initialisieren (Aufgabe des Betriebssystems) enthält.

Bibliotheken der C- (und C++) Standardfunktionen werden vom Compiler-Hersteller bereitgestellt. In Fortran sind Standardfunktionen in die Sprache integriert. Häufig verwendete Funktionen können auch vom Anwender zu eigenen Bibliotheken zusammengefaßt werden.
Optional steuert das `Makefile` (über das make-Programm) das Compilieren und Linken der einzelnen Teile des Programms. Es ist insbesondere bei größeren Projekten von Vorteil, weil automatisch nur jene Module neu übersetzt werden, die einer Änderung unterzogen wurden.

1.1.1 Ein einfaches FORTRAN-Programm

```
PROGRAM HALLO
WRITE(*,*) 'Hallo Freunde!'
END
```

Die erste Zeile ist optional und legt den Namen des Programms fest: `HALLO`.
Der `WRITE`-Befehl enthält in Klammern Angaben über das Ausgabemedium und das Ausgabeformat.
Der erste Stern steht für die default-Ausgabe, den Bildschirm.
Der zweite Stern bezeichnet das Defaultformat.
Die auszugebende Zeichenkette steht hier in einfachen Anführungszeichen.

Sie können dieses kleine Testprogramm ausführen, indem sie eine Datei erzeugen, die zur Kennzeichnung, daß es sich um ein Fortran-Programm handelt, günstigerweise mit `.f` endet, z.B. `xyz.f`. Dann können Sie den Befehl
`f77 -o xyz.out xyz.f ;./ xyz.out`
ausführen.

Fortran-Programme müssen folgendes Format für den Programmtext beachten:
Alle Anweisungen müssen (irgendwo) innerhalb der Schreibstellen 7 bis 72 einer Zeile stehen. Zur Klarstellung sind hier die Spaltennummern angegeben:

```
0       1       2       3       4       5       6       7
12345678901234567890123456789012345678901234567890123456789012
```

```
PROGRAM HALLO
WRITE(*,*) 'Hallo Freunde!'
END
```

Kommentare werden durch die Zeichen C oder *, die in der ersten Spalte stehen müssen, gekennzeichnet:

```
C Das ist eine Kommentar
* Einen Kommentar kann man nach dem Fortran77-Standard auch so schreiben
! Manche Compiler, wie der Gnu-Fortran77-Compiler, f77,
!       erlauben auch solche Kommentare

* Auch die vorige Zeile, eine Leerzeile, ist als Kommentar erlaubt.
```

Der Fortran77-Standard ist nachzulesen unter:
`http://www.fortran.com/fortran/F77_std/rjcnf0001.html`.

Die Beschreibung des Gnu-Fortran f77-Compilers ist verfügbar unter:
`http://world.std.com/ burley/g77.html/index.html#toc_Language`.
Die Beschreibung von Kommentarzeilen im Fortran77-Standard steht unter:
`http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-3.html#sh-3.2.1`,
Spezielles zu Kommentaren in g77 stehen in
`http://world.std.com/ burley/g77.html/`
`Statements-Comments-Lines.html#Statements%20Comments%20Lines`

Ist der Text einer Zeile zu lang, um in die Schreibstellen 7 bis 72 zu passen, kann er in der nächsten fortgesetzt werden, wenn diese als Fortsetzungszeile gekennzeichnet ist. Die Kennzeichnung besteht aus irgendeinem Zeichen, außer “0” und “ “ (hier ein “&”) in Spalte 6:

```
0         1         2         3         4         5         6         7
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

```
      WRITE(*,*)
     & 'Hallo Freunde!'
      END
```

Moderne Fortran-Versionen erlauben meist folgende Vereinfachungen:

1. Schlüsselwörter können nach Belieben in Groß- und Kleinbuchstaben geschrieben werden.
2. Der Tabulator als erstes Zeichen einer Zeile wird als Sprung in die 7.Spalte interpretiert, außer
3. Der Tabulator als erstes Zeichen gefolgt von einer Ziffer wird als Fortsetzungszeile interpretiert.

1.1.2 Der Aufbau eines FORTRAN-Programmes

Komplexe Fortranprogramme bestehen aus einem einzigen Hauptprogramm und beliebig vielen Unterprogrammen. Die Anordnung dieser Programmteile ist beliebig. Ob es sich bei einem Programmteil um ein Hauptprogramm oder um ein Unterprogramm handelt ist an der ersten Anweisung zu erkennen. Unterprogramme beginnen mit dem Schlüsselwort **SUBROUTINE**, Funktionsunterprogramme mit **FUNCTION**, **REAL FUNCTION** oder **INTEGER FUNCTION**.

```
C     Die folgende Zeile ist optional
      PROGRAM HALLO
*     es gibt in diesem einfachen Programm nur einen Ausfuehrungsteil
*     springe ins Unterprogramm
      CALL AUSGABE('Hallo Freunde')
      END

*     das ist ein Unterprogramm
      SUBROUTINE AUSGABE(TEXT)
*     es folgt der Vereinbarungsteil
      CHARACTER*13 TEXT
```

```
*       es folgt der Ausfuehrungsteil
        WRITE(*,*) TEXT
*       springe zurueck ins Hauptprogramm
        RETURN
        END
```

Fortranprogramme beginnen mit der Titelzeile, dann kommt ein Vereinbarungsteil, in dem die Variablen und Felder definiert werden, daran schließt der Ausführungsteil, der mit dem Schlüsselbefehl **END** endet.

2 Einfache Sprachelemente

2.1 Einfache Zuweisungen und Objekte

Speicherplätze, die Zahlenwerte (also ein Bitmuster) aufnehmen können, werden durch Namen (Variablen) angesprochen. Sie sind damit einfache Objekte. Um das Bitmuster korrekt interpretieren zu können, muß der Objekt-Typ angegeben werden, das heißt, die Variable wird ”deklariert”. Einfache Datentypen sind beispielsweise ganze Zahlen (integer) und Gleitkommazahlen (real/float).

C und C++:

```
int main()
{

    int myint;       /*integer*/
    float myfloat; /*floating point*/

    myint=100;
    myfloat=3.14;

    printf("%i␣%f\n",myint,myfloat);
}
```

FORTRAN:

```
PROGRAM Explizite Deklaration

IMPLICIT NONE

INTEGER myint
REAL myfloat

myint=100
myfloat=3.14

WRITE(*,*) myint, myfloat
END
```

Die Deklaration in FORTRAN kann auf zwei Arten erfolgen. Als Default werden alle Variablen, deren Namen mit einem der Buchstaben i,j,k,l,m,n beginnt, automatisch als **INTEGER** deklariert, alle anderen als **REAL**. Dies kann durch explizite Deklarationen für einzelne Variablen verändert werden, ohne das Defaultverhalten generell zu ändern. Die Deklaration **IMPLICIT** NONE setzt das Defaultverhalten außer Kraft.

```
PROGRAM Implizite Deklaration

imyint=100
fmyfloat=3.14

WRITE(*,*) imyint, fmyfloat

END
```

C und C++:
Groß/Kleinschreibung wird in C und C++ beachtet (`Banane` und `bAnane` sind zwei verschiedene Variablen).
Die maximale Länge des Variablennamens ist compilerabhängig und muß zur Erfüllung des Standards in C/C++ mindestens 31 Zeichen sein.

FORTRAN:
Variablennamen bestehen in Fortran aus einem bis 6 Zeichen, von denen der erste ein Buchstabe sein muß, die weiteren Zeichen dürfen auch Ziffern sein, z.B. E605.
Groß- und Kleinbuchstaben werden nicht unterschieden.
Für den g77 dürfen die Variablennamen beliebig lang sein.

FORTRAN:
In Fortran können Anweisungen eine eindeutige Adresse, ein Label, bekommen, das aus fünf Ziffern bestehen kann.
Solche Adressen können in den Spalten 1 bis 5 angeordnet werden, wobei führende Nullen und Leerzeichen bedeutungslos sind.

```
      program Label
      do 10 i=1,3
   10 print*, 'i=',i
      end
```

2.2 Einfache Felder = Vektoren = Arrays

2.2.1 Syntax

Vektoren enthalten mehrere (im Speicher aneinandergereihte) Objekte eines Typs. Mit Hilfe des Index' kann auf die einzelnen Elemente zugegriffen werden.

C und C++:
Fünf `int`-Variablen: `iA[5]`
Hier sind die Inhalte der Speicherstellen, die symbolisch mit `iA[0],iA[1],...` bezeichnet wurden, dargestellt.

57	7	34	1234	-678
`iA[0]`	`iA[1]`	`iA[2]`	`iA[3]`	`iA[4]`

Statische Feldvereinbarungen erfolgen nach dem Vorbild normaler Typvereinbarungen, wobei die Feldgröße durch eine Konstante festgelegt wird (der Compiler kann sie auch aus der Anzahl der Initialisierungselemente selbst bestimmen). Jedenfalls muß die Größe des Felds zum Zeitpunkt des Compilierens festgelegt sein. Der Compiler reserviert den Speicherplatz (= Größe eines Elementes (in bytes) * Anzahl der Elemente).
Dynamische Feldvereinbarungen sind ebenfalls möglich.
Genauere Erklärungen dazu folgen im Kapitel über **Zeiger** (*pointer*).

Der Feldindex wird in eckigen Klammern angegeben und ist vom Typ int

Man kann beliebige berechenbare Ausdrücke angeben, wenn diese durch automatische Standardkonversion in den Typ int umgewandelt werden können.

Der Index [0] bezeichnet das erste Feldelement

Beachten Sie, daß daher bei einer Dimensionierung [N] der Feldindex maximal den Wert [N-1] annehmen darf. Die Dimensionierung beschreibt die Anzahl der Feldelemente, nicht den maximalen Indexwert.

```
/* Vereinbarung */
/* 100 int-Feldelemente */
int iArray[100];
int i;

/* Durchlaufen des Arrays */
/* Zuweisung von Werten */
for(i=0;i<100;i++)iArray[i]=i*2;

/* Ausgabe */
for(i=0;i<100;i++)printf("iA[%d]:%d\n",i,iArray[i]);
```

Erzeugt int-Array mit 100 Elementen. Belegt das Array mit Werten (i*2). Druckt die Werte samt Index wieder aus.

FORTRAN:

Feldvereinbarung

Felder werden im Vereinbarungsteil eines Fortranprogammes, der vor dem Ausführungsteil kommt, festgelegt.
vek1 hat die 2 Elemente vek1(1) und vek1(2).
vek2 enthält vek2(−1), vek2(0), vek2(1).
ivek hat 5 Elemente.
name hat 2 Elemente mit je 6 Buchstaben.

```
      Program Feldvereinbarung

C Vereinbarungsteil
      Real vek1(2),vek2(-1:1)
      Integer ivek(5)
      Character*6 name(2)
C Ausfuehrungsteil
```

flexible Feldvereinbarung

Die Feldvereinbarung erfolgt in Fortran 77 immer statisch. Sie kann jedoch durch die Einführung von Parametern flexibel gestaltet werden.

```
      Program Flexible Felder
C Vereinbarungsteil
      Implicit None
      Integer laenge,indu,indo
      Parameter(laenge=7,indu=-3,indo=3)
      Real array(laenge),feld(indu:indo)
C Ausfuehrungsteil
```

Indizierung von Feldelementen
Felder werden mit runden Klammern indiziert, genauso wie die Felddeklarationen erfolgen.

```
      Program Feldindizierung
C Vereinbarungsteil
      Parameter(laenge=10)
      Integer feld(laenge)
C Ausfuehrungsteil
      do i=1,laenge
*        hier eine Festlegung von Feldelementen
         feld(i)=i**2
*        hier eine Ausgabe einzelner Feldelemente
         Write(*,*) i,"-Quadrat=",feld(i)
      enddo

*     ein Ausdruck des ganzen Feldes
      Write(*,*) feld
*     ein Ausdruck der Teilliste der geraden Zahlen
      Write(*,*) (feld(k),k=2,10,2)
      END
```

Wie das obere Beispiel zeigt, kann die Ein- und Ausgabe von Feldelementen auf drei Arten erfolgen: (1) mit einzelnen Feldelementen, (2) als gesamter Vektor oder (3) mit einer sogenannten “impliziten Do-Schleife”.

Die Einhaltung der Feldgrenzen wird weder vom Compiler noch zur Laufzeit geprüft!
Das Überschreiten des vereinbarten Speicherbereichs ist eine der tückischesten Fehlerquellen in der Fortran und C-Programmierung. Zur Fehlervermeidung gibt es jedoch Compileroptionen mit denen eine Feldüberschreitung zur Laufzeit laufend überprüft werden kann.

2.2.2 Initialisieren von Arrays

Ein Array kann durch Definition initialisiert werden:

```
int iDown[10]={10,9,8,7,6,5,4,3,2,1};
float v[3]={3.33,4.44,5.55};
```

in solchen Fällen kann der Compiler selbst die Felddimensionierung vornehmen:

```
int iDown[]={10,9,8,7,6,5,4,3,2,1};
float v[]={3.33,4.44,5.55};
```

FORTRAN:

In den bisherigen Fortran-Beispielen haben wir Variablen und Feldelementen im Ausführungsteil, also zur Programmlaufzeit festgelegt. Es gibt jedoch mit der **DATA**-Anweisung die Möglichkeit schon vor der Programmausführung festzulegen.

```
      Program data_Anweisung
C Vereinbarungsteil
      implicit none

      integer feld(3)
      real zeit(7)
      character*10 tag(7)

      data feld/1,2,3/,zeit/7*0./
      data tag/'Montag','Dienstag','Mittwoch',
     *         'Donnerstag','Freitag','Samstag','Sonntag'/
```

2.2.3 Felder von Zeichen

Einen eigenständigen Datentyp ”String” gibt es in C nicht.

Ein String ist ein Array von Zeichen

Da es keine Funktion gibt, die das Ende eines Felds erkennen kann, wird das Ende einer Zeichenkette mit dem Zeichen `\0` (=ASCII-Zeichen Null) symbolisiert. Viele Funktionen bearbeiten Zeichenketten nur bis zum ersten auftretenden `\0` oder schließen ein Ergebnis (neuen String) damit ab. Bei der Größenvereinbarung ist der erforderliche Platz für das Zeichen `\0` zu berücksichtigen.

2.2.4 Mehrdimensionale Arrays

Beispiel für ein initialisiertes, zweidimensionales Array:

```
int iUpandDown[2][10]=
  {  {10,9,8,7,6,5,4,3,2,1},
     {1,2,3,4,5,6,7,8,9,10}
  };
```

Die Indexfolge entspricht hier [Zeile][Spalte] ([row][column]). Die inneren Klammern unterstreichen die Struktur, sind aber nicht erforderlich:

```
int iUpandDown[2][10]=
  {10,9,8,7,6,5,4,3,2,1,1,2,3,4,5,6,7,8,9,10};
```

Bei der Dimensionierung kann der erste Index weggelassen werden

```
int iUpandDown[][10]=
  {10,9,8,7,6,5,4,3,2,1,1,2,3,4,5,6,7,8,9,10};
```

Felder von Zeichen:

```
char cSchirm[25][80];
```

ist ein Array von Arrays, das 25 Arrays vom Typ char mit je 80 Elementen (= 2000 Zeichen = Bytes) speichern kann.

FORTRAN:

Die Dimensionen in mehrdimensionalen Feldern werden durch einen Beistrich getrennt feld2d(0:7,−3:3), genauso wie die Indizes eines Matrixelementes

```
      Program data_Anweisung
C Vereinbarungsteil
      integer n,mn
      parameter(n=5,mn=((n-1)*n)/2)

*     Die Feldgroesse kann man auch mit DIMENSION festlegen
      dimension feld2d(n,n)
      data ((feld2d(i,j),i=1,j-1),j=2,n)/mn*1./
      data (feld2d(i,i),i=1,n)/n*0./
      data ((feld2d(j,i),i=1,j-1),j=2,n)/mn*-1./

C Ausfuehrungsteil
      write(*,*) 'eine antisymmetrische Matrix'
      do i=1,n
         Write(*,'(10f5.1)') (feld2d(j,i),j=1,n)
      enddo
      END
```

Deklaration und Initialisierung von Strings

`'\0'` wird automatisch angehängt

```
char szString[9]="Beispiel"; /* [9] für 8 Zeichen plus '\0' */
/* oder */
char szString[]="Beispiel";  /* Compiler berechnet Länge */
```

Auf ein einzelnes Zeichen wird einfach zugegriffen:

```
szString[0]='B'; szString[4]='p'; szString[8]=0;
```

Beachten Sie die unterschiedliche Verwendung von einfachen und doppelten Anführungszeichen:

`'A'` ... Ein Zeichen, ASCII-Code von A also eine Integerzahl.
`"A"` oder `"ABCd"` ... Eine Zeichenkette, ein Feld mit abschließendem `\0`-Zeichen.

Es ist **nicht** möglich, einem String außerhalb der Deklaration durch Zuweisung einen Wert zu geben:

```
char szS[9];             /* nicht initialisierter String */
char szT[9]="Beispiel";  /* OK in der Deklaration */
szS="Beispiel";          /* Fehler, Zuweisung außerhalb der Deklar. */
szS=szT;                 /* Geht auch nicht (siehe "pointer") */
```

Für das Kopieren von Strings gibt es einige Bibliotheksfunktionen. Einhaltung der vereinbarten Feldgrenzen beachten!

2.3 Beenden eines Programms

Normalerweise endet ein Programm nach der letzten Anweisung in `main()`, danach übernimmt wieder das Betriebssystem die Kontrolle. Man kann das Programm auch mit der Funktion `exit()` anhalten (z.B. in tief geschachtelten Funktionen). Dadurch wird ein (wählbarer) Wert an das Betriebssystem übergeben, den man (sofern das Betriebssystem es vorsieht) auswerten kann (z.B. in einer BATCH-Datei; siehe Standardbibliothek).

2.4 Variablenübergabe in Fortran-Programmen

Wie schon erwähnt bestehen Fortranprogramme aus einem einzigen Hauptprogramm und den Unterprogrammen.
Ein Unterprogramm dient im allgemeinen zur Berechnung irgendwelcher Zwischenresultaten aus irgendwelchen Eingabedaten. Dazu ist es notwendig, die Eingabedaten an das Unterprogramm zu übergeben und die Ausgabedaten wieder an das übergeordnete Programm zurückzugeben.
Es ist erlaubt die Eingabedaten zu verändern und damit den veränderten Wert an das übergeordnete Programm zurückzugeben.
Die Variablenübergabe geschieht sehr einfach in der Variablenliste im Unterprogrammaufruf *CALL* und in der Unterprogrammtitelzeile **SUBROUTINE**.

Es ist unbedingt notwendig, daß die Zahl und Typen der Variablen im *CALL*- und im **SUBROUTINE**-Befehl übereinstimmen!

Die Variablennamen der übergebenen Variablen müssen im Unterprogramm und im übergeordneten Programm nicht übereinstimmen. Man nennt das die Argumentassoziierung, siehe auch Abschnitt 3.1.6.
Variablen eines Unterprogrammes, die nicht explizit übergeben werden, sind lokale Variable.

In folgenden Beispielen werden Variablen auf drei Arten übergeben:

- im Unterprogrammaufruf: R und A
- im Funktionsnamen: KUGVOL
- im Common PIE die Variablen PI und E

```
************************************************************************
      PROGRAM Unterprogramm_Aufruf
************************************************************************
      WRITE(*,*) 'Bitte Kreisradius R eingeben:'
      READ(*,*) R
      IF(R.GT.15.) STOP 'R ist zu gross !'
      CALL KREIS(R,A)
      WRITE(*,'("Kreis: R=",F8.3,", A=",F8.3)') R,A
      END

************************************************************************
*     das ist ein Unterprogramm zur Berechnung der Kreisflaeche
************************************************************************
      SUBROUTINE KREIS(R,F)
*     PI ist eine lokale Variable
      PI=4.*ATAN(1.)
*     Berechnung der globalen Ausgabevariablen
      F=PI*R*R
*     der folgende Befehl ist unbedingt notwendig
      END

************************************************************************
      PROGRAM Funktionsunterprogramm
************************************************************************

      WRITE(*,*) 'Bitte Kreisradius R eingeben:'
      READ(*,*) R
      IF(R.GT.15.) STOP 'R ist zu gross !'

      V=KUGVOL(R)
      WRITE(*,'("Kugel: R=",F8.3,", V=",F8.3)') R,V
      END

************************************************************************
*     das ist ein Funktionsunterprogramm
************************************************************************
      FUNCTION KUGVOL(R)
*     PI ist eine lokale Variable
      PI=4.*ATAN(1.)
*     Berechnung des Funktionswertes
      KUGVOL=4./3.*PI*R**3.
      END
```

```
*********************************************************************
      PROGRAM Common_Beispiel
*********************************************************************
*     die globale Variable PI steht im COMMON-Bereich /PIE/
      COMMON /PIE/PI,E
      PI=4.*ATAN(1.)

      WRITE(*,*) 'Bitte Kreisradius R eingeben:'
      READ(*,*) R
      V=KUGVOL(R)
      WRITE(*,'("Kugel: R=",F8.3,", V=",F8.3)') R,V
      PRINT*, 'auch E ist nun bekannt, E=',E
      END

*********************************************************************
      FUNCTION KUGVOL(R)
*********************************************************************
      COMMON /PIE/PI,E
* eventuell Ruecksprung ins Hauptprogramm
      IF(R.LE.0.) RETURN
*     Berechnung des gesuchten Funktionswertes
      KUGVOL=4./3.*PI*R**3.
      E=EXP(1.)
      END
```

Zahl und Typen der Variablen im COMMON müssen unbedingt übereinstimmen!

Variablen-, Felddeklarationen und *COMMON*s reservieren Speicherplätze einer gewissen Größe. Innerhalb einer Programmeinheit, sind diesen Speicherplätzen mit ihrem Namen anzusprechen. In einer anderen Programmeinheit kann jedoch derselbe Speicherplatz mit einem anderen Namen identifiziert werden. Welcher Name in einem Unterprogramm welchem Speicherplatz zugeordnet ist, entscheidet die Variablenliste im Unterprogrammaufruf und die Definition des *COMMON*-Bereiches. Deshalb sind diese Variablenliste sehr sorgfältig zu kontrollieren. Es handelt sich hier um eine der häufigtsten, manchmal auch schwierig zu lokalisierenden, Fehlerquellen.
Neben dem benannten *COMMON*-Bereichen, kann man auch einen unbenannten *COMMON*-Bereich definieren.

```
* das ist ein unbenannter COMMON-bereich fuer die Variablen PI und E
      COMMON PI,E
```

Haupt- und Unter-Programme enden in Fortran, wie schon erwähnt, mit einer eigenen **END**-Zeile.

Die Ausführung des Programmes kann an einer beliebigen Stelle mit einem **STOP**-Befehl, wie im obigen Beispiel, beendet werden.
Falls ein Programm mehrere **STOP**-Befehle enthält, weil das Programm aus unterschiedlichen Gründen angehalten werden soll, ist es günstig einen Text, der das **STOP** charakterisiert, auszugeben.

3 Sprachelemente in FORTRAN

3.1 Datentypen und Wertebereiche in FORTRAN

3.1.1 FORTRAN Zeichensatz

Der Fortran-Zeichensatz besteht aus 26 Buchstaben, 10 Ziffern und 13 Sonderzeichen, siehe auch:
`http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-3.html#sh-3.1`

- Buchstaben
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- Ziffern
  0 1 2 3 4 5 6 7 8 9
- Sonderzeichen

Zeichen	Zeichenname
	Leerzeichen
=	Gleichheitszeichen
+	Plus
-	Minus
	Stern
/	Schrägstrich
(	Linke Klammer
)	Rechte Klammer
.	Dezimalpunkt
,	Beistrich
$	Dollar
’	Apostroph
:	Doppelpunkt

3.1.2 Datentypen

Es gibt 6 Datentypen, siehe auch
`http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-4.html#sh-4`

- Integer
- Real
- Double precision

- Complex
- Logical
- Character

Den Datentyp für symbolische Namen, Konstanten, Variablen, Felder, externe Funktionen, und Funktionsunterprogramme kann man in Typenzeilen festlegen.
Falls eine Typenangabe fehlt, ist der Datentyp von symbolischen Namen, die mit I, J, K, L, M, oder N beginnen **INTEGER**, sonst **REAL**.
Diese automatische Typzuordnung kann durch das **IMPLICIT**-Befehl abgeändert werden.

```
************************************************************************
      PROGRAM doppeltgenau
************************************************************************
*     (A–H,O–Z) sollen doppeltgenaue Variablen charakterisieren
      IMPLICIT DOUBLE PRECISION (A–H,O–Z)

      PI=4.D0*ATAN(1.D0)
      WRITE(*,10) PI
 10   FORMAT('PI ist hier auf 13 Stellen genau, PI=',F17.15)
      END
```

3.1.3 Konstanten

Die f77-Standards zu Konstanten finden Sie unter:
`http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-4.html#sh-4.2`
Spezialitäten von f77 unter:
`http://world.std.com/ burley/g77.html/Constants.html#Constants.`

Hier nur einige Beispiele zur Festlegung von Konstanten.

```
************************************************************************
      PROGRAM Arithmetische_Variable
************************************************************************
      REAL A,B
      DOUBLE PRECISION C,D,G
      COMPLEX E,F

*     zwei reelle Konstante
      A=4.E–2;     B=4.E–02
*     zwei doppelt genaue Konstante
      C=4.D–2;     D=4.D–02
      WRITE(*,'(2E30.20)') A,B,C,D
*     zwei komplexe Konstante
      E=(–0.1,2); F=(0.,1.)
      WRITE(*,*) E,F
*     Addition zweier Zahlen mit verschiedener Byteanzahl
      G=A+C
```

```
      WRITE(*,'(2E30.20)') G
      END

************************************************************************
      PROGRAM CHARACTER_LOGICAL
************************************************************************
*     Der IMPLICIT-Befehl gehoert vor den speziellen Befehl
      IMPLICIT LOGICAL (L)
      CHARACTER*31 TEXT

      TEXT='Ein Buchstabe braucht ein Byte'
      WRITE(*,*) TEXT
      L1=.TRUE.
      L2=.FALSE.
      WRITE(*,*) 'Logische Variable schreibt man mit Punkten an!'
      WRITE(*,*) L1; WRITE(*,*) L2
      END
```

3.1.4 Symbolische Konstanten

Mit dem Befehl #define kann man symbolische Konstanten definieren, die allerdings vor der Übersetzung durch Aufruf des Präprozessors durch den tatsächlichen Wert ersetzt werden. Das klingt vielleicht kompliziert, ist es jedoch nicht, wie das folgende Beispiel zeigt. Erzeugen Sie ein Programm `test.f`

```
************************************************************************
      PROGRAM define
************************************************************************
#  define A 20.
      PRINT,* A
      END
```

und führen Sie g77 -x f77-cpp-input -o test.out test.f ; ./test.out aus.
Die Option -x f77-cpp-input ruft den Präprozessor auf und ersetzt die Konstante überall im Programm durch ihren Wert.

Vielleicht wundern Sie sich über das Ergebnis in

```
************************************************************************
      PROGRAM Rechenfehler?
************************************************************************
#     define N 1+1
      PRINT*, N*N
      END
```

3.1.5 Wertebereich der Datentypen

Der Wertebereich der Datentypen INTEGER, REAL, COMPLEX, CHARACTER ist von der Maschine abhängig. Der Standard ist festgelegt in:
`http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-2.html#sh-2.13`.

Die meisten Compiler erlauben eine Notation, wie sie Standard-Fortran für *CHARACTER* erlaubt, *CHARACTER*$*n$, z.B. **REAL**$*$4,**INTEGER**$*$2,**LOGICAL**$*$1. Die Variable n gibt hier i.a. die Byte-Anzahl an, siehe auch
http://world.std.com/ burley/g77.html/
Star-Notation.html#Star%20Notation
und
http://world.std.com/ burley/g77.html/
Double-Notation.html#Double%20Notation.
Die Genauigkeit ergibt sich dann aus aus der Bit-Anzahl. Für **INTEGER**-Variablen ist ein Bit für das Vorzeichen nötig, für Real-Variable ist neben dem Vorzeichen noch die Aufteilung zwischen Mantissen-Bits und Exponenten-Bits zu berücksichtigen.

3.1.6 Assoziierung

Siehe auch
http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-2.html#sh-2.14.

Darunter versteht man, daß ein und derselbe Speicherplatz mit unterschiedlichen symbolischen Namen angesprochen wird. Dies betrifft

1. Argument-Assoziierung
2. COMMON
3. EQUIVALENCE
4. ENTRY

Unter Argumentassoziierung versteht man, daß die symbolischen Namen von Variablen in einem *CALL*-Befehl nicht mit den symbolischen Namen des **SUBROUTINE**- oder **FUNCTION**-Befehls übereinstimmen müssen, siehe auch Abschnitt 2.4.
Wie ebenso bereits in Abschnitt 2.4 besprochen, können die Speicherplätze eines *COMMON*-Bereiches von unterschiedlichen Unterprogrammen aus mit unterschiedlichen symbolischen Namen angesprochen werden.
Sowohl im Falle der Argument-Assoziierung, als auch im Falle der *COMMON*-Assoziierung ist jedoch genauestens darauf zu achten, daß die Variablenanzahl und Variablentypen in den beiden einander entsprechenden Listen exakt übereinstimmen.
Mit dem **EQUIVALENCE**-Schlüsselwort kann derselbe Speicherbereich in derselben Programmeinheit mit unterschiedlichen symbolischen Namen angesprochen werden). Im folgenden Beispiel belegen die Felder MATRIX und VEKTOR denselben Speicher. Beachten Sie im folgenden Beispiel die Abbildung der Matrix auf den Vektor. Der Schreibbefehl zeigt, daß der erste Index eines mehrdimensionalen Feldes schneller läuft als der zweite.

```
*********************************************************************
      PROGRAM Equivalence
*********************************************************************
      INTEGER MATRIX(2,2),VEKTOR(4)
      EQUIVALENCE (MATRIX(1,1),VEKTOR(1))

      DO I=1,4
         VEKTOR(I)=I
      ENDDO

      WRITE(*,'(2i3)') (MATRIX(1,I),I=1,2),(MATRIX(2,I),I=1,2)
      PRINT*, 'Achten Sie auf die Belegung der Matrixelemente !'
      END

*********************************************************************
      PROGRAM Entry
*********************************************************************
      WRITE(*,*) 'Bitte radius R eingeben:'
      READ(*,*) R
      CALL KREIS(R,A)
      WRITE(*,'("Kreis:␣R=",F8.3,",␣A=",F8.3)') R,A
      CALL KUGEL(R,A,V)
      WRITE(*,'("Kugel:␣R=",F8.3,",␣A=",F8.3,",␣V=",F10.3)') R,A,V
      END

*********************************************************************
      SUBROUTINE KREIS(R,F)
*********************************************************************
      PI=4.*ATAN(1.)
      F=PI*R*R
      RETURN
      ENTRY KUGEL(R,F,V)
      F=4.*PI*R*R
      V=4.*PI/3.*R**3.
      END
```

3.2 Ausdrücke in FORTRAN

Man unterscheidet zwischen:

1. Arithmetischen Ausdrücken
2. Vergleichsausdrücke
3. Logischen Ausdrücken

3.2.1 Arithmetische Ausdr¨ucke und Operatoren

Siehe `http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-6.1`.
Die arithmetischen Operatoren wirken zwischen zwei Operanden

**	Potenzierung
/	Division
	Multiplikation
-	Subtraktion oder Vorzeichen
+	Addition oder Vorzeichen

oder $+$ und $-$ als Vorzeichen nur auf einen Operanden.
Für die Bildung arithmetischer Ausdrücke gelten folgende Regeln:

1. Weder zwei Operatoren, noch zwei Operanden dürfen unmittelbar aufeinander folgen.
2. Das Multiplikationszeichen muß stets ausgeschrieben werden.
3. Undefinierte arithmetische Ausdrücke sind verboten.

falscher Ausdruck	richtiger Ausdruck
A/-B	A/(-B)
A**-2	A**(-2)
2N+1	2*N+1

Die Prioritätenfolge für die Auswertung arithmetischer Ausdrücke lautet mit sinkender Priorität:

1. Klammerausdrücke
2. Potenzierung
3. Multiplikation und Division
4. Addition und Subtraktion

Potenzen werden nach der Regel Term$**$Exponent ausgewertet, d.h. $2**3**2$ bedeutet $2**(3**2)$.
Der Datentyp arithmetischer Ausdrücke hängt von den Datentypen der beteiligten Operanden ab.
Es gelten dabei die folgenden Regeln:

1. Eine Verknüpfung eines **INTEGER** und eines **REAL**-Termes ergibt einen **REAL**-Ausdruck. Vor der Ausführung der Operation wird dabei – mit Ausnahme der ganzzahligen Potenzierung – der Wert des **INTEGER**-Operanden in den entsprechende **REAL**-Ausdruck umgewandelt.

2. Für eine ganzzahlige Division ist zu beachten, daß das Ergebnis der ganzzahlige Anteil des Quotienten ist. Der Rest geht verloren, es wird also nicht gerundet.

3. In einer Potenzierung mit negativ ganzzahligem Exponenten wird über den Kehrwert berechnet. Es ergibt sich demnach $3 * *(-2) = 0$

In einem **REAL**-Ausdruck sind zur Sicherheit alle ganzen Zahlen als **REAL**-Zahlen darzustellen. Neben der Fehlervermeidung werden dadurch auch unnotwendige Zahlenumwandlungen vermieden.

```
************************************************************************
      PROGRAM Fehlerquellen_in_Operationen
************************************************************************
      PI=4.*ATAN(1.)
      PRINT*, 'Beachte folgende Fehlerquellen'
      PRINT*, '1/100=',1/100
      PRINT*, '5**(1/2)=',5**(1/2)
      PRINT*, '1/3*4*PI=',1/3*4*PI
      PRINT*, '4*PI/3=',4*PI/3
      PRINT*, '3**(-2)=',3**(-2)
      PRINT*, '3.**(-2)=',3.**(-2)
      PRINT*, '3**(-2.)=',3**(-2.)
      END
```

Hier ein Beispiel mit Verwendung der Abschneidefunktion

```
************************************************************************
      PROGRAM un_gerade
************************************************************************
      DO I=1,10
         IF ((I/2)*2.EQ.I) THEN
            PRINT*, I,' ist eine   gerade Zahl'
         ELSE
            PRINT*, I,' ist eine ungerade Zahl'
         ENDIF
      END DO
      END
```

3.2.2 Operationen mit Zeichenketten

Siehe `http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-6.2`.
CHARACTER-Konstante werden in Apostrophen angeschrieben.
Innerhalb von Zeichenketten auftretende Leerzeichen sind signifikant.

Jeder innerhalb einer Zeichenkette auftretende Apostroph muß durch zwei Apostrophe dargestellt werden.
Die Längenangabe einer Zeichenkette kann auch hinter dem Namen erfolgen und gilt nur für diese Größe.
Hinter dem Schlüsselwort *CHARACTER* angegebene Längen gelten für alle Variablen ohne eigene Längenangabe.
In Längen von in **PARAMETER**-Anweisungen festgelegten *CHARACTER*-Konstanten können auch automatisch bestimmt werden.
Mit // kann man Zeichenketten zusammenhängen und mit : kann man Teile von Zeichenketten herausgreifen.

```
*****************************************************************
      PROGRAM Zeichenketten
*****************************************************************
      CHARACTER*(*) LEER,DAS,IST,EIN,MEISTER*7
      CHARACTER*10 APOSTROPH
      PARAMETER (LEER=' ',DAS='Das',IST='ist',EIN='ein')
      MEISTER='mEIster'
      APOSTROPH='APO''STROPH'
      PRINT*, 'Der Satz '
      PRINT*, DAS,LEER,IST,LEER,EIN,LEER,
     &        APOSTROPH(4:4),MEISTER(2:3),APOSTROPH(4:4)
      LAENGE=LEN(DAS//LEER//IST//LEER//EIN//LEER//
     &        APOSTROPH(4:4)//MEISTER(2:3)//APOSTROPH(4:4))
      WRITE(*,*) 'besteht aus ',LAENGE,' Buchstaben'
      END
```

3.2.3 Vergleichsausdrücke

Siehe `http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-6.3`.
Die Vergleichsoperatoren lauten

Operator	Bedeutung
.LT.	Less than
.LE.	Less than or equal to
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to

Die logischen Operatoren lauten mit abfallender Priorität:

Operator	Bedeutung
.NOT.	nicht
.AND.	und
.OR.	inklusives oder
.EQV. or .NEQV.	gleich oder ungleich

Hier ein Beispiel zu logischen Operatoren und Vergleichsausdrücken und ihrer Priorität:

```
************************************************************************
      PROGRAM Logical
************************************************************************
      IMPLICIT LOGICAL (L)
      DATA L1/.TRUE./,L2/.FALSE./,L3/.FALSE./
      PRINT*, 2.GT.1 .OR. 2.LE.1 .AND. 4.LE.3
      PRINT*, (2.GT.1 .OR. 2.LE.1) .AND. 4.LE.3
      PRINT*
      PRINT*, (L1 .OR. L2 .AND. L3) .NEQV. (L1 .OR. L2) .AND. L3
      PRINT*, (L1 .OR. L2 .AND. L3) .EQV. L1 .OR. (L2 .AND. L3)
      END
```

3.2.4 Prioritäten und Verknüpfungsreihenfolge

Siehe http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-6.5.
Die Priorität der verschiedenen Operationen lautet in abfallender Reihenfolge

Operator	Priorität
Arithmetisch	hoch
Zeichen	
Vergleich	
Logisch	nieder

Die Reihenfolge für die Verknüpfung von Termen ist wie folgt festgelegt:

1. Klammerausdrücke
2. Priorität der Operationen
3. Rechts nach links Interpretation der Potenzierung
4. Links nach rechts Interpretation von Multiplikationen und Divisionen
5. Links nach rechts Interpretation von Additionen und Subtraktionen
6. Links nach rechts Interpretation der Verknüpfung von Zeichenketten
7. Links nach rechts Interpretation von Verknüpfungen in logischen Termen
8. Links nach rechts Interpretation von logischen Äquivalenzen

3.3 Anweisungen in FORTRAN

Man unterscheidet zwischen ausführbaren und nicht ausführbaren Anweisungen, siehe
http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-7.html#sh-7.1

Ausführbare Anweisungen sind:

1. Zuordnungen
2. GO TO
3. IF, ELSE IF, ELSE und END IF
4. CONTINUE
5. STOP und PAUSE
6. DO
7. READ, WRITE und PRINT
8. REWIND, BACKSPACE, ENDFILE, OPEN, CLOSE und INQUIRE
9. CALL und RETURN
10. END

Nicht ausführbare Anweisungen sind:

1. PROGRAM, FUNCTION, SUBROUTINE, ENTRY und BLOCK DATA
2. DIMENSION, COMMON, EQUIVALENCE, IMPLICIT, PARAMETER, EXTERNAL, INTRINSIC und SAVE
3. INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL und CHARACTER
4. DATA
5. FORMAT
6. Funktionsanweisung

3.4 Steueranweisungen

Siehe http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-11.html#sh-11
Es gibt 16 Anweisungen zur Kontrolle des Programmflusses:

1. Unbedingtes **GO TO**
2. Berechnetes **GO TO**
3. Zugeordnetes **GO TO**
4. Arithmetisches **IF**
5. Logisches **IF**
6. - **Block IF**
7. - **ELSE IF**
8. - **ELSE**
9. - **END IF**
10. - **DO**
11. - *CONTINUE*
12. - **STOP**
13. - **PAUSE**
14. - **END**
15. - *CALL*
16. - **RETURN**

```
************************************************************************
      PROGRAM unbedingte_GOTOs
************************************************************************
      ISUM=0
 100  PRINT*, 'Bitte eine ganze Zahl eingeben, STOP bei 0'
      READ(*,*) I
      ISUM=ISUM+I
      IF(I.EQ.0) GOTO 200
      GOTO 100
 200  WRITE(*,*) 'Die Summe äbetrgt ',ISUM
      END
```

Es ist nicht erlaubt in eine Schleife hineinzuspringen.

```
************************************************************************
      PROGRAM berechnetes_GOTO
************************************************************************
      PRINT*, 'Bitte eine ganze Zahl eingeben, STOP bei 0'
 10   READ(*,*) I
      IF(I.EQ.0) GO TO 40

      GO TO (20,30) MOD(I,2)+1
 20   PRINT*, I,' ist eine gerade Zahl'
      GO TO 10
 30   PRINT*, I,' ist eine ungerade Zahl'
      GO TO 10

 40   END
```

Das zugeordnete GOTO-Anweisung und das arithmetische **IF**
http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-11.3
http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-11.4
werden kaum verwendet. Sehr wichtig ist jedoch das logische **IF**, wie sie im letzten Beispiel verwendet wurde: **IF**(I.EQ.0) **GO TO** 40. Wenn sich der Vergleich I.EQ.0 als wahr herausstellt, wird die nachfolgende Anweisung **GO TO** 40 ausgeführt.
Sollen nach Erfüllung der Bedingung mehrere Anweisungen ausgeführt werden, so muß man ein **Block IF** verwenden, das aus **IF** ... **THEN** besteht und alle Anweisungen bis zum nächsten **ELSE IF**, **ELSE**, oder **END IF** ausführt.

```
************************************************************************
      PROGRAM Block_IF
************************************************************************
      N=0;  ISUM=0
 100  PRINT*, 'Bitte eine ganze Zahl eingeben, STOP bei 0'
      READ(*,*) I
      IF(I.EQ.0) GOTO 200
      N=N+1;  ISUM=ISUM+I
      GOTO 100
 200  IF(N.NE.0) THEN
         AVE=FLOAT(ISUM)/FLOAT(N)
         WRITE(*,*) 'Der Mittelwert äbetrgt ',AVE
      ENDIF
      END
```

```
************************************************************************
      PROGRAM ELSEIF
************************************************************************
 100  PRINT*, 'Bitte eine Zahl eingeben'
      READ(*,*) A
      IF(A.LT.0) THEN
         WRITE(*,*) 'Das Vorzeichen ist negativ'
      ELSE IF(A.GT.0) THEN
         WRITE(*,*) 'Das Vorzeichen ist positiv'
      ELSE
         WRITE(*,*) 'Das Vorzeichen ist nicht bestimmbar'
      END IF
      END
```

Nach Standardfortran, siehe:
http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-6.html#sh-11.3
haben Schleifen an einem LABEL zu enden, wie im folgenden Beispiel. Statt einer wirklichen Anweisung kann das LABEL auch an der leeren Anweisung *CONTINUE* stehen.

```
************************************************************************
      PROGRAM DO
************************************************************************
      DO 100 I=1,3
         PRINT*, I
 100  CONTINUE
      END
```

Aber praktisch alle Übersetzer erlauben statt dieser Schleifenform die **DO** ... **END DO**-Konstruktion, die auch schon in den bisherigen Beispielen verwendet wurde, siehe z.B. die g77-Beschreibung in
http://world.std.com/ burley/g77.html/END-DO.html#END%20DO.
Die erste Zeile der **DO**-Schleife hat dann die Form **DO** laufende_Zahl=Anfangswert,Endwert,Abstand. Die Variable laufende_Zahl kann dabei vom Typ **INTEGER**, **REAL** oder **DOUBLE PRECISION** sein, ebenso der Anfangswert, der Endwert oder der Abstand. Diese drei Zahlen werden zur Berechnung des aktuellen Wertes der Laufvariable in deren Typ umgewandelt.

```
************************************************************************
      PROGRAM Indexkonversion
************************************************************************
      A=1.; B=5.;  C=1.9
      S=A
      DO I=A,B,C
         PRINT*, 'Index=',I,' ,reeller Wert waere :',S
         S=S+C
      END DO
      END
```

Der Abstandswert in der Schleife darf nicht 0 sein; er kann jedoch entfallen, dann wird er automatisch mit 1 angenommen. Negative Abstandswerte sind erlaubt.

```
************************************************************************
      PROGRAM Rueckwaertsschleife
************************************************************************
      A=5.; B=1.;  C=-1.9
      DO D=A,B,C
         PRINT*, D
      END DO
      END
```

Die **DO** ... WHILE-Schleife gibt es in Fortran77-Standard nicht, die meisten Compiler können sie jedoch ohne Schwierigkeiten verarbeiten, siehe auch
http://world.std.com/ burley/g77.html/DO-WHILE.html#DO%20WHILE

```
************************************************************************
      PROGRAM  DO_WHILE
************************************************************************
      I=5
      DO WHILE (I.GT.0)
         PRINT*, I
         I=I-1
      END DO
      END
```

Auch das **DO** forever können die meisten Übersetzer verarbeiten, siehe auch
http://world.std.com/ burley/g77.html/DO-WHILE.html#DO%20WHILE

```
************************************************************************
      PROGRAM do_forever
************************************************************************
      I=0
      DO
         I=I+1
         PRINT*, I
         IF(I.GT.10) STOP
      END DO
      END
```

Die Anweisungen **STOP**, **END** und **RETURN** haben wir schon mehrfach verwendet. Die **PAUSE**-Anweisung wird nicht mehr benötigt.

3.5 Nicht ausführbare Anweisungen in FORTRAN

Die Anweisungen **PROGRAM**, **FUNCTION**, **SUBROUTINE** und **ENTRY** haben wir bereits besprochen. Die **BLOCK DATA**-Anweisung wird benötigt um Variable in benannten *COMMON*-Blöcken zu initialisieren. Variablen im unbenannten *COMMON*-Block dürfen nicht initialisiert werden. Näheres zur **BLOCK DATA**-Anweisung, siehe in
`http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-16`
`http://world.std.com/ burley/g77.html/Block-Data-and-Libraries.html#Block%20Data%20and%20Libraries`

```
*****************************************************************
      PROGRAM BLOCK_DATA
*****************************************************************
      COMMON /BENAMT/A
      PRINT*, A
      END

      BLOCK DATA blodat
      COMMON /BENAMT/A
      DATA A /21./
      END
```

Die Anweisungen **DIMENSION**, *COMMON*, **EQUIVALENCE**, **IMPLICIT**, **PARAMETER** wurden schon erklärt. Nun zu **EXTERNAL**, **INTRINSIC** und **SAVE**.
EXTERNAL und **INTRINSIC** müssen verwendet werden, wenn Funktionsnamen, wie im folgenden Beispiel FKT und SIN an der Stelle von symbolischen Variablen verwendet werden, wie im Unterprogrammaufruf *CALL SUBR(X,SIN)*, siehe auch: `http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-8.7`

```
*****************************************************************
      PROGRAM External_Intrinsic
*****************************************************************
      INTRINSIC SIN; EXTERNAL FKT
      X=0.01
      CALL SUBR(X,FKT); CALL SUBR(X,SIN)
      END

      SUBROUTINE SUBR(X,F)
      PRINT*, F(X)
      END

      FUNCTION FKT(X)
      X=SIN(X)/X
      END
```

Wenn ein Unterprogramm verlassen wird, gehen die Werte der lokalen Variablen verloren. Wenn man dennoch mit den Werten von Variablen beim nächsten Unterprogrammaufruf weiterarbeiten möchte, so kann man dazu die **SAVE**-Anweisung benützen. Als Argumente der **SAVE**-Anweisung sind lokale Variable und Felder, sowie Namen von *COMMON*-Blocks erlaubt. Die Namen von *COMMON*-Blocks muß man durch Schägstriche begrenzen.

```
************************************************************************
      PROGRAM SAVE
************************************************************************
      DO I=1,5
         CALL SUBR(0.1)
      END DO
      END

      SUBROUTINE SUBR(DX)
      SAVE S
      DATA S/1./
      S=S-DX
      PRINT*,S
      END
```

3.6 Anweisungsfunktionen und interne Funktionen

Eine Anweisungs-Funktion wird in einer einzigen Anweisung definiert. Sie weist einen arithmetischen, logischen oder zeichenartigen Wert zu. Eine Anweisungsfunktion kann nur in der Programmeinheit verwendet werden, in der sie definiert wird. Anweisungsfunktionen müssen in ihrer Programmeinheit am Ende des Vereinbarungsteiles stehen.
Anweisungsfunktionen sind definiert in:
`http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-15.4`.

```
************************************************************************
      PROGRAM AnweisungsFUNKTION
************************************************************************
      RUNDE(A,N)= ANINT(A*10.**N)/10.**N

      PRINT*, 'Bitte die Zahl der Kommastellen eingeben'
      READ(*,*) N
      DO WHILE (Z.NE.0)
         PRINT*, 'Bitte die Zahl eingeben'
         READ(*,*) Z
         WRITE(*,*) RUNDE(Z,N)
      END DO
      END
```

Die internen Funktionen sind definiert in:
http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-15.10.

Intrinsic Function	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Function
Conversion to Integer int(a)	1	INT	- INT IFIX IDINT -	Integer Real Real Double Complex	Integer Integer Integer Integer Integer
Conversion to Real	1	REAL	REAL FLOAT - SNGL	Integer Integer Real Double	Real Real Real Real
Conversion to Double	1	DBLE	- - - -	Integer Real Double Complex	Double Double Double Double
Conversion to Complex	1 or 2	CMPLX	- - -	Integer Real Double	Complex Complex Complex
Conversion to Integer	1		ICHAR	Character	Integer
Conversion to Character	1		CHAR	Integer	Character

Intrinsic Function	Generic Name	Specific Name	Type of Argument	Type of Function
Truncation	AINT	AINT DINT	Real Double	Real Double
Nearest Whole Number	ANINT	ANINT DNINT	Real Double	Real Double
Nearest Integer	NINT	NINT IDNINT	Real Double	Integer Integer
Absolute Value	ABS	IABS ABS DABS CABS	Integer Real Double Complex	Integer Real Double Real

Intrinsic Function	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Function
Remaindering	2	MOD	MOD AMOD DMOD	Integer Real Double	Integer Real Double
Transfer of Sign	2	SIGN	ISIGN SIGN	Integer Real	Integer Real
Positive Difference	2	DIM	IDIM DIM DDIM	Integer Real Double	Integer Real Double
Double Precision Product	2		DPROD	Real	Double
Choosing Largest Value	$>= 2$	Max	MAX0 AMAX1 DMAX1	Integer Real Double	Integer Real Double
			AMAX0 MAX1	Integer Real	Real Integer
Choosing Smallest Value	$>= 2$	MIN	MIN0 AMIN1 DMIN1	Integer Real Double	Integer Real Double
			AMIN0 MIN1	Integer Real	Real Integer
Length	1		LEN	Character	Integer
Substringindex	2		INDEX	Character	Integer
Imaginary Part	1		AIMAG	Complex	Real
Conjugate	1		CONJG	Complex	Complex
Square Root	1	SQRT	SQRT DSQRT CSQRT	Real Double Complex	Real Double Complex

Intrinsic Function	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Function
exp(a)	1	EXP	EXP DEXP CEXP	Real Double Complex	Real Double Complex
ln(a)	1	LOG	ALOG DLOG CLOG	Real Double Complex	Real Double Complex
log10(a)	1	LOG10	ALOG10 DLOG10	Real Double	Real Double
sin(a)	1	SIN	SIN DSIN CSIN	Real Double Complex	Real Double Complex
cos(a)	1	COS	COS DCOS CCOS	Real Double Complex	Real Double Complex
tan(a)	1	TAN	TAN DTAN	Real Double	Real Double
arcsin(a)	1	ASIN	ASIN DASIN	Real Double	Real Double
arccos(a)	1	ACOS	ACOS DACOS	Real Double	Real Double
arctan(a)	1	ATAN	ATAN DATAN	Real Double	Real Double
arctan(a1/a2)	2	ATAN2	ATAN2 DATAN2	Real Double	Real Double
sinh(a)	1	SINH	SINH DSINH	Real Double	Real Double
cosh(a)	1	COSH	COSH DCOSH	Real Double	Real Double
tanh(a)	1	TANH	TANH DTANH	Real Double	Real Double

Intrinsic Function	Number of Arguments	Specific Name	Type of Argument	Type of Function
Lexically Greater Than or Equal	2	LGE	Character	Logical
Lexically Greater Than	2	LGT	Character	Logical
Lexically Less Than or Equal	2	LLE	Character	Logical
Lexically Less Than	2	LLT	Character	Logical

3.7 Ein/Ausgabe in FORTRAN

3.7.1 Ein/Ausgabe-Befehle

In den bisherigen Fortranbeispielen haben wir schon vielfach Lese- und Druckbefehle verwendet. Hier werden diese Befehle nun etwas systematischer behandelt. Die Definition des f77-Standards finden Sie in
`http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-12`.
Es gibt in Fortran folgende neun Ein/Ausgabe-Befehle:

1. READ
2. WRITE
3. PRINT
4. OPEN
5. CLOSE
6. INQUIRE
7. BACKSPACE
8. ENDFILE
9. REWIND

3.7.2 Read, Write und Print

Der einfachste Lesebefehl lautet **READ** $*$, variablenliste. Dieser enthält den Stern zur Angabe des vom System vorgegebenen Formates f.
Die möglichen Formen der Lese- und Schreibbefehle lauten:

- READ f [,iolist]
- PRINT f [,iolist]
- READ (cilist) [iolist]
- WRITE (cilist) [iolist]

iolist ist wie in den bisherigen Beispielen eine Folge von symbolischen Variablennamen, Feldnamen oder Zeichenketten.
cilist steht für “command information list” und kann folgende Informationen enthalten:(cilist)=
=([**UNIT** =] u , [**FMT** =] f , **REC** = rn, **IOSTAT** = ios, **ERR** = s, **END** = s).
cilist muß zumindest eine Angabe über Lese- bzw. Schreibeinheit enthalten. Wird ein Stern statt der Nummer u der Schreib- und Leseeinheit eingesetzt, so wird als

Ausgabe- bzw. Eingabeeinheit, die vom System vordefinierte Einheit verwendet, das ist bei interaktiver Verarbeitung üblicherweise das Terminalfenster. Durch den **OPEN**-Befehl können andere Dateien als Lese- oder Schreibeinheit aufgemacht werden. Die eckigen Klammern in [**UNIT** =]u und in [**FMT** =] f bedeuten, daß diese Schlüsselwörter weggelassen werden können.
u ist eine positive ganze Zahl, die einer Datei zugeordnet ist, die im **OPEN**-Befehl festgelegt wird.
f ist eine Formatangabe in einer der folgenden Formen:

1. *, für das Format, das vom System vorgegeben ist,
2. eine Anweisungsnummer, die sich auf eine Format-Anweisung bezieht,
3. eine Zeichenkette, die in einer runden Klammer die Formatanweisung enthält.

Die Formatanweisung wird etwas später in größerem Detail beschrieben. Hier nun einige Beispiele für Schreib- und Leseanweisungen.

```
************************************************************************
      PROGRAM IO-Befehle
************************************************************************
*     ein * fuer das Format, das vom System vorgegebenen ist
*     und automatische Einheitsvorgabe
      PRINT*, 'Bitte eine ganze Zahl eingeben'
*     Leseanweisung auf Standard-Leseeinheit 5 (Terminal)
      READ(UNIT=5,FMT=*) N
*     Schreibanweisung auf Standard-Druckeinheit 6 (Terminal)
      WRITE(UNIT=6,FMT=*) 'Sie haben die Zahl',N,' eingegeben!'
      WRITE(6,*) 'Die Ausgabe ist im Standardformat erfolgt.'
*     Eine Ausgabe mit Angabe einer Formatnummer
      WRITE(*,9999) FLOAT(N)/2.
 9999 FORMAT('Die Haelfte Ihrer Zahl lautet',E9.2)
*     Dieselbe Ausgabe äerhlt man mit einer Zeichenkette,
*     die in einer runden Klammer die Formatanweisung äenthlt
      WRITE(*,'("Die Haelfte Ihrer Zahl lautet",E9.2)') FLOAT(N)/2.
      END
```

Die Formatangabe [**FMT** =] f in einem Lese- oder Schreibbefehl entfällt bei unformatierter Ausgabe, siehe folgendes Beispiel:

```
************************************************************************
      PROGRAM unformatierte Ein_Ausgabe
************************************************************************
      I=5
*     unformatierte Ausgabe auf die Einheit 1. Da der Einheit 1 kein
*     Dateiname zugeordnet ist, wird auf fort.1 ausgeschrieben
      WRITE(1) I
      I=0
*     unformatiertes Einlesen von Einheit 1, d.h. von fort.1
      REWIND (1)
      READ(1) I
      WRITE(*,*) 'Die Zahl I lautet wieder, I=',I
      END
```

Wird beim Einlesen das Dateiende festgestellt, so kann man mit **END** = s zur Anweisung mit der Nummer s springen.

```
************************************************************************
      PROGRAM Ende_der_Daten
************************************************************************
      I=5
      WRITE(1) I
      REWIND (1)
      READ(1) I
      READ(1,END=10) I

      WRITE(*,*) 'Die Zahl I lautet wieder, I=',I
      STOP
 10   PRINT*, 'Nicht existente Daten kann man nicht einlesen !'
      END
```

Tritt beim Einlesen ein Fehler auf, so kann man mit **ERR** = s entsprechende Anweisungen zur Fehlerbehandlung ansteuern.

```
************************************************************************
      PROGRAM Lesefehler
************************************************************************
      I=5
      WRITE(1) I
      REWIND (1)
*     Ein Fehler beim Einlesen eines Records kann speziell behandelt werden
      READ(1,ERR=10) I,I

      WRITE(*,*) 'Die Zahl I lautet wieder, I=',I
      STOP
 10   PRINT*, 'Das angeforderte Record ist zu lang !'
      END
```

Mit **IOSTAT** = ios kann man die Integer-Variable ios mit folgenden Werten über den Input-Output-Status belegen

- ios< 0 für Lesen des Dateiendes
- ios$= 0$ für fehlerfreies Lesen
- ios> 0 im Falle eines Lesefehlers enthält die Variable ios die Nummer des Fehlers.

Die Bedeutung der Fehlernummern kann man unter
`http://world.std.com/ burley/g77.html/Run-time-Library-Errors.html#Run-time%20Library%20Errors`
nachlesen.

```
************************************************************************
      PROGRAM iostat
************************************************************************
      DOUBLE PRECISION D
      I=5
      WRITE(1) I
      REWIND (1)
*     Eingabe-Zustand in Variable NUMR ablegen
      READ(1,IOSTAT=NUMR) R
*     Das Record ist lang genug, die Variable R ist aber unsinnig
      PRINT*, 'NUMR=',NUMR,', R=',R
      REWIND (1)
      READ(1,IOSTAT=NUMR) D
      PRINT*, 'Die Fehlernummer fuer ein zu kurzes Record lautet ',NUMR
      END
```

3.7.3 Open, Close und Inquire

Siehe http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-12.10. Die **OPEN**-Anweisung dient dazu um einem Dateinamen eine Einheitennummer eineindeutig zuzuordnen. Diese Einheitennumm kann dann in einem **READ**, **WRITE** oder **PRINT**-Befehl verwendet werden, um diese Datei anzusprechen. Wenn noch keine Datei mit diesem Namen existiert, so wird durch das **OPEN** eine solche Datei erzeugt.

Die **OPEN**-Anweisung hat die Form:
OPEN (olist) wobei olist eine Einheitsnummer u enthalten muß und die übrigen der folgenden Parameter optional sind:

- [UNIT =] u
- ERR = s
- FILE = dateiname
- STATUS = sta
- ACCESS = acc
- FORM = fm
- RECL = rl
- BLANK = blnk

dateiname ist eine Zeichenkette, die einen erlaubten Namen einer Datei enthält.
Der Status kann lauten: **STATUS** = 'OLD', 'NEW', 'SCRATCH', 'UNKNOWN'. Für **STATUS** = 'SCRATCH' darf **FILE** = dateiname nicht angegeben werden. 'UNKNOWN' ist der default Wert.
Der Default Access-Parameter ist **ACCESS** = '**SEQUENTIAL**', also eine sequentiell, wie ein Magnetband, abzuarbeitende Datei. **ACCESS** = '**DIRECT**' erlaubt es die einzelnen

Records in beliebiger Reihenfolge anzusprechen. Die Recordnummer wird im **READ** oder **WRITE**-Befehl durch den Parameter **REC** festgelegt.
Der Parameter **RECL** = rl darf nicht für eine sequentielle Datei angegeben werden, sondern nur für eine Direct-Access-Datei. Alle Records müssen dieselbe Länge rl besitzen, die meist in Bytes gemessen wird.

```
*****************************************************************
      PROGRAM DirectAccess
*****************************************************************
      A=5.; B=2.
*     ein Direct-access-record wird ausgeschrieben,
*     die Form ist default unformatiert,
*     pro Real-Variable braucht es eine Recordlaenge von 4 (Bytes)
      OPEN(UNIT=1,ACCESS='DIRECT',RECL=4)
      WRITE(1,REC=3) A
      WRITE(1,REC=5) B
      READ(1,REC=3,IOSTAT=IOS) B
      PRINT*, IOS,B
      END
```

Der Defaultwert für **FORM** ist für eine sequentielle Datei **FORM** = '**FORMATTED**' und für eine Direct-Access-Datei **FORM** = '**UNFORMATTED**'.
Der Parameter **BLANK** darf nur für formatierte Dateien verwendet werden. Sein Defaultwert ist **BLANK** = 'NULL'. Er wird nur sehr selten verwendet, seine Beschreibung steht in
Siehe `http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-12.10`.

Die allgemeine Form der *CLOSE*-Anweisung lautet *CLOSE (cllist)*, wobei *c llist* eine Einheitsnummer u enthalten muß und die übrigen der folgenden Parameter optional sind:

- [**UNIT** =] u
- **IOSTAT** = ios
- **ERR** = s
- **STATUS** = sta

Der **STATUS**-Parameter kann den Defaultwert **STATUS** = 'KEEP' annehmen oder **STATUS** = 'DELETE' lauten.
Der **INQUIRE**-Befehl gibt nähere Auskünfte über Einheiten, bzw. die ihnen zugeordneten Dateien. Eine nähere Beschreibung ist aus `http://www.fortran.com/fortran/F77_std/jcnf0001-sh-16.html#sh-12.10.3` zu entnehmen.

3.7.4 Rewind, Backspace und Endfile

Diese Anweisungen dienen zur Positionierung in einer sequentiellen Datei. Ihre Form lautet:

- **BACKSPACE** u

- **BACKSPACE** (alist)
- ENDFILE u
- ENDFILE (alist)
- **REWIND** u
- **REWIND** (alist)

alist muß den Parameter [**UNIT** =] u enthalten. Weiters können die Parameter **IOSTAT** = ios, **ERR** = s angegeben werden.
Mit dem **BACKSPACE**-Befehl geht man ein Record zurück. Die Anweisung ENDFILE erzeugt eine EndOfFile-Marke.

```
************************************************************************
      PROGRAM backspace
************************************************************************
      DIMENSION B(3); DATA A/3.14/,B/1.,2.,3./
      OPEN(1,FILE='fort.dat',FORM='UNFORMATTED')
*     schreibe zwei Records
      WRITE(99) A; WRITE(99) B
*     gehe an den Beginn des zweiten Records zurueck
      BACKSPACE 99
*     die erste Zahl des zweiten Records wird nun eingelesen
      READ(99) A
      PRINT*, 'A=',A
      END

************************************************************************
      PROGRAM EndOfFile
************************************************************************
      DIMENSION B(3); DATA A/3.14/,B/1.,2.,3./
      OPEN(99,FILE='fort.dat',FORM='UNFORMATTED')
      WRITE(99) A; WRITE(99) B
      ENDFILE 99
*     gehe an den Beginn der EndOfFile-Marke
      BACKSPACE 99
*     ueberschreibe die EndOfFile-Marke
      WRITE(99) A
*     gehe an den Beginn des letzten Records
      BACKSPACE 99
*     die erste Zahl des letzten Records wird nun eingelesen
      READ(99) B(1)
      PRINT*, 'B(1)=',B(1)
      END
```

3.7.5 Format-Anweisungen

Fortran enthält vielfältige und komfortable Formatbeschreibungen,siehe
`http://www.fortran.com/fortran/F77_std/rjcnf0001-sh-13.html#sh-13`
Die Form der Formatanweisung lautet:

labl **FORMAT** fs

Die Formatanweisung braucht ein LABEL labl, weil sie sonst nicht in einer Eingabe/Ausgabe-Anweisung angegeben werden kann.
Die Formatspezifikation fs ist eine in runden Klammern und durch Beistriche getrennte Liste von Formatbeschreibern (Edit Descriptoren).
Die folgenden Formatbeschreiber sind wiederholbar:

1. `Iw`, eine rechtsbündige **INTEGER**-Zahl aus w Zeichen,
2. `Iw.m`, wie Iw, aber mindestens m Ziffern werden ausgegeben, nötigenfalls führende Nullen,
3. `Fw.d`, Festpunktdarstellung mit d Nachkommastellen für **REAL**, **DOUBLE PRECISION** und *COMPLEX*-Zahlen,
4. `Ew.d`, Gleitkommadarstellung mit einer Mantisse zwischen 0.1 und 1.0 und d Nachkommastellen,
5. `Ew.dEe`, wie oben mit einem Exponenten aus e Stellen,
6. `Dw.d`, wie Ew.d,
7. `Gw.d`, je nach der Größe der Zahl erfolgt die Ausgabe im E oder F Format, die Ausgabe von Sternen wird hier vermieden,
8. `Gw.dEe`, wie oben mit einem Exponenten aus e Stellen,
9. `Lw`, in einem Feld der Länge w wird entweder T für logische Variable, oder F ausgegeben,
10. `A`, die volle Zeichenkette entsprechend ihrer Definition,
11. `Aw` , Zeichenkette in einem Feld der Länge w, ist das Feld zu kurz, werden nur die ersten w Zeichen ausgegeben, ist es zu lang, werden die ersten Stellen mit Leerzeichen gefüllt.

w steht hier für die Feldweite. Die Wiederholungsanzahl steht als ganze Zahl vor diesen Formatbeschreibern, siehe folgendes Beispiel.
Falls die Ausgabe nicht in das vorgegebene Format paßt, werden Sterne ausgedruckt.
Hier einige Beispiele:

```
************************************************************************
      PROGRAM Format_fuer_Dezimalzahlen
************************************************************************
      REAL R(3); DATA R/1.,2.,3./
*     Eine Format-Anweisung braucht ein Label
 11   FORMAT('3F5.2-Format:',3F5.2,
     &    ' mit 5 Zeichen Breite und 2 Nachkommastellen')
      WRITE(6,11) R
      WRITE(6,12) R
```

```
      WRITE(6,13) R
      WRITE(6,14) R
      WRITE(6,15) 0.01*R(2),0.1*R(2),R(2),10.*R(2),100.*R(2)
 12   FORMAT('3E8.1-Format:',3E8.1)
 13   FORMAT('3E8.1E1-Format:',3E8.1E1)
 14   FORMAT('3D8.1-Format:',3D8.1)
 15   FORMAT('4G8.1,G9.3-Format:',4G8.1,G9.3)
      END

*****************************************************************
      PROGRAM Format_fuer_Integer_Character_Logisch
*****************************************************************
      INTEGER I(3);  CHARACTER*3 TEXT;  LOGICAL L
      DATA I/1,2,3/,TEXT/'ABC'/,L/.TRUE./
 10   FORMAT('3I5-Format:',3I5)
 11   FORMAT('3I5.3-Format:',3I5.3)
      WRITE(6,10) I
      WRITE(6,11) I
      WRITE(6,'(A,'','',A2,'','',A5)') TEXT,TEXT,TEXT
      WRITE(6,'(''L5-Format:'',L5)') L
      END
```

Nicht wiederholbare Formatbeschreiber (Steuerungsbeschreiber) sind:

1. `'...'`, ein Text unter Apostrophen wird als Zeichenkette ausgegeben, in einer **READ**-Anweisung ist dieser Formatbeschreiber nicht erlaubt,
2. `nH`, die `n` Zeichen nach dem `H` werden als Zeichenkette ausgegeben, auch dieser Formatbeschreiber ist in einer **READ**-Anweisung nicht erlaubt,
3. `Tc`, rücke weiter zu Position *c*,
4. `TLc`, gehe *c* Zeichen zurück,
5. `TRc`, gehe *c* Zeichen weiter,
6. `nX`, gehe `n` Zeichen weiter,
7. `/`, geht zum Beginn des nächsten Records, der nächsten Zeile,
8. `:`, beendet die Ausgabe, außer es sind noch nicht alle Daten ausgeschrieben,
9. `S`, die Ausgabe eines positiven Vorzeichens ist von der Rechenanlage abhängig,
10. `SP`, ein positives Vorzeichen wird explizit ausgeschrieben,
11. `SS`, ein positives Vorzeichen wird nicht ausgegeben,
12. `kP`, reelle Werte werden skaliert: für einen F-Beschreiber in der Ausgabe findet eine Skalierung mit 10^k statt, für eine Eingabezahl ohne Exponent eine Skalierung mit 10^{-k}, für Ein- und Ausgaben von Zahlen mit Exponent wird nur die Mantisse um 10^k bzw. 10^{-k} verschoben und der Exponent entsprechend korrigiert,

13. BN, Leerstellen in der Eingabe werden ignoriert,

14. BZ , Leerstellen in der Eingabe werden als Nullen interpretiert.

```
************************************************************************
      PROGRAM Steuerungsbeschreiber
************************************************************************
      I=1234567890
*     ein H-Formatbeschreiber
      WRITE(6,'(5Hxxxxy)')
*     ein T-Formatbeschreiber
      WRITE(6,'(''xxx'',T5''xxx'',T6''yyy'',T80''z'',
     &          TL2''Z'',TL20,''hier'',TR1,''dort'',1X,''wo?'')')
*     ein /-Formatbeschreiber beginnt eine neues Record (neue Zeile)
      WRITE(6,'(''eine Zeile'',/,''noch eine Zeile'')')
*     ein :-Formatbeschreiber beendet die Ausgabe
      WRITE(6,'(''eine dritte Zeile'',:,'' eine vierte Zeile ?'')')
* ß    auer es ßmu noch eine Ausgabe erfolgen
      WRITE(6,'(''eine vierte Zeile'',:,'', I='',I10)') I
      END

************************************************************************
      PROGRAM weitere_Steureungsbeschreiber
************************************************************************
      REAL R(3);  DATA R/1.1111,2.2222,3.3333/
*     Steuerung der weiteren Vorzeichenausgabe durch SP, SS, S
      WRITE(6,'(SP,F5.0,F5.0,S,F5.0)') R
*     Skalierung in Ausgabe
      WRITE(1,'(3(3PF7.1))') R
*     ohne Skalierung einlesen
      REWIND 1
      READ(1,'(3F7.1)') R
*     Daten sind veraendert
      WRITE(6,'(3(4PE15.4))') R
      END
```