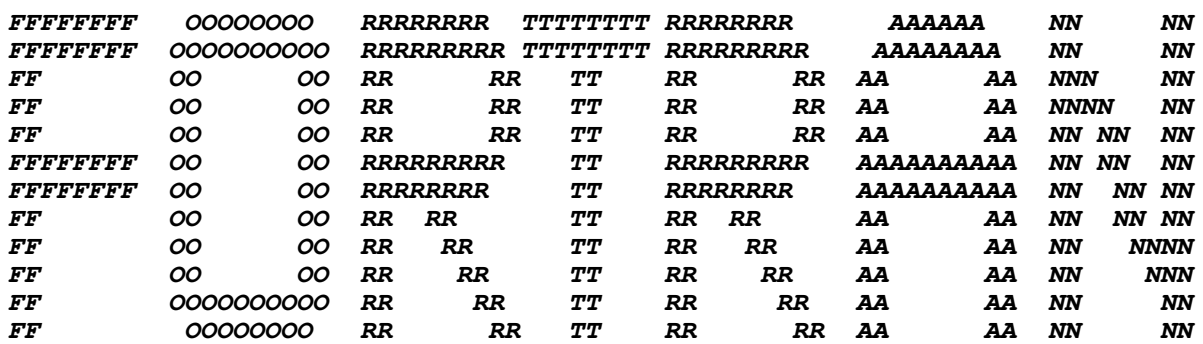
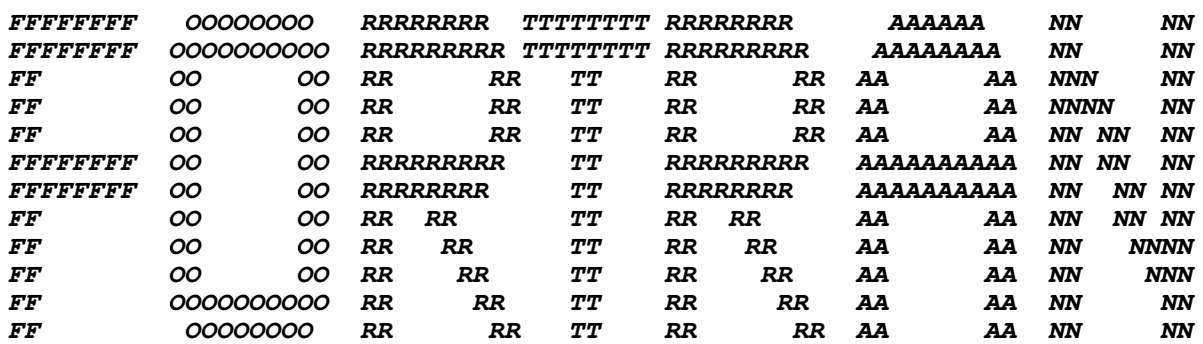


© S.Vetter URZ Heidelberg 1986,1994,1997



© S.Vetter URZ Heidelberg 1986,1994,1997

- 3 -

Vorbemerkungen

Dieses Skriptum ist so formatiert, daß es als HTML-File gleichzeitig mit
lpr -Pf68umps
als Postscript-File mit korrektem Seitenumbruch ausgedruckt werden kann, z.B. mit Netscape Print... mit Paper Size A4.

Besondere Schreibweise innerhalb des Skripts:

DOS-Kommandos, Editor-Kommandos, Debug-Kommandos:
Ist ein Kommando groß, gegen Ende jedoch kleingeschrieben, so stellt der großgeschriebene Teil die Minialabkürzung dar, muß also mindestens soweit geschrieben werden. Im übrigen sind unter DOS alle Kommandos unempfindlich gegen Groß / Kleinschreibung.

Kursiv geschriebene Angaben müssen durch den entspr. Wert ersetzt werden.

- 4 -

SKRIPTUM
zu Fortran77

Inhaltsverzeichnis

Kapitel 0: Einleitung **07**

A. WATFOR unter WEDIT (PC) **07**
1. Schritt: Workspace bekommt einen Namen (oder Laden File) 07
2. Schritt: Schreiben / Editieren des Programms 07
3. Schritt: Ausführen des Programms 08
4. Schritt: Ergebnisse / Programmlisting 08
5. Schritt: Wie 1. oder 2. oder Editor verlassen 08
Hilfen für Fortranprogramme am PC 09
Ausdrucken eines Files 09
Die Kommandos zum Filehandling (Diskette/Festplatte) 09

B. Wie werden Fortran77-Programme geschrieben? **10**

Kapitel 1: Mensch - Computer - Programm **11**

Was kann der Computer? 11
Warum benutzen wir überhaupt Computer? 11
5 Schritte zur Lösung eines Problems mit dem Computer 12
Erste Beispiele 12
Übungen 14
Was passiert, wenn man einen Fehler macht? 14
Übungen 17
Programmierstil 17
Zusammenfassung 17
Aufgaben 19

Kapitel 2: Zahlen, Arithmetik und Variablen **20**

Ganze Zahlen (INTEGER) 20
Übungen 20
Zahlen mit Dezimalpunkt (REAL) 20
Konstante, Variable 21
Erklärung des Typs (Deklaration) 22
Zuweisung von Werten an Variable 22
Einlesen von Werten 22
Umlenken der Standard-Ein/Ausgabe 24
Eingebaute Funktionen 25
3 Fehlerquellen bei der Arithmetik mit REAL-Zahlen 25
Aufgaben 26

- 5 -

Kapitel 3: Entscheidungen und Steueranweisungen **28**

Logische Ausdrücke 28
Der IF-Block 28
Übungen 31
Logische Operatoren 32
Aufgaben 32

Kapitel 4: Schleifen **34**

Schleifen und Schleifensteuerung 34
WHILE-Schleife 34
Übungen 34
DO-Schleife 35
Übungen 37
Mehrere Endebedingungen in der Schleife 37
Geschachtelte Schleifen 38
Aufgaben 39

Kapitel 5: Felder **41**

Übungen 42
Ein- und Ausgabe von Feldern 43
Übungen 43
2-dimensionale Felder - Tabellen 43
Übungen 45
Höher dimensionierte Felder 45
Angabe einer unteren Indexgrenze 45
Zuweisen von Anfangswerten 46
Zusammenfassung 46
Aufgaben 47
Noch einmal: Schleifensteuerung; der Typ LOGICAL 48

Kapitel 6: Unterprogramme **50**

Programmeinheiten 50
Subroutine-Unterprogramme 50
Übungen 53
Function-Unterprogramme 54
Übung 54
Ein weiteres Beispiel 55
Module, Programmbibliotheken 57
Aufgaben 57
Dimensionierung zur Ausführungszeit 59
Externe Unterprogramme 60
Zusammenfassung: Wirkungsweise des Aufrufs 61
Gültigkeitsbereich von Namen 62
Aufgaben (Fortsetzung) 62

Kapitel 7: Ein- und Ausgabe **63**

Eingabe 63
Formen der READ-Anweisung 63
Formatfreie Eingabe 64
Einlesen einer unbekannten Anzahl von Daten 64
Formatierte Eingabe: Die Grundgedanken 66
Übungen 66
Ausgabe 67
Formen der WRITE-Anweisung 68
Formatfreie Ausgabe 68
Formatierte Ausgabe 68
Ausgabe von INTEGER-Zahlen 68
Der I-Code 68
Der T-Code 69
Übungen 69
Der X-Code 69
Implizite Angabe der Ausgabe-Position 70
ASA-Druckersteuerzeichen 70
Übungen 70
Ausgabe von REAL-Zahlen 71
Der F-Code 71
Der E-Code 71
Der D-Code 71
Der G-Code 72
Übersicht über alle Formatcodes 72
Wiederholungsfaktoren, Code-Gruppen 74
Übungen 74
Aufgaben 75
Zusammenfassung: Allgemeine Regeln für Formate 75
Tabellarische Übersicht 76

Kapitel 8: Der Typ CHARACTER **78**

CHARACTER-Werte 78
Konstante, Variable, Ausdrücke 78
Eingebaute Funktionen zur Stringverarbeitung 79
Übungsbeispiel 79
Aufgaben 81
Interne Files 82
Ein letztes Beispiel 83

Anhang A: Kurzbeschreibung der Syntax von Fortran **87**

Anhang B: Literatur zu Fortran **89**

Anhang C: Der WATFOR Debugger (PC) **90**

Anhang D: Lösungen zu den Übungsaufgaben **91**
zu Kap. 1 91
zu Kap. 2 93
zu Kap. 3 95
zu Kap. 4 98
zu Kap. 5 103
zu Kap. 6 107
zu Kap. 7 113

Index **116**

Kapitel 0: Einleitung

A. WATFOR unter WEDIT (PC)

Falls noch nicht geschehen, schalten Sie Computer und Monitor ein.
Sonst starten Sie das System ohne Aus- und Einschalten und drücken
Strg, **Alt** und **Entf** gleichzeitig.
Im Laufwerk darf auf keinen Fall eine Diskette stecken!

Falls Sie mit Diskette arbeiten wollen, legen Sie diese in das entspr. Laufwerk. Verwenden Sie die Diskette erstmalig, geben Sie das Kommando
FORMAT A: und Return-Taste für 3 1/4 Zoll Disketten bzw.
FORMAT B: und Return-Taste für 5 1/2 Zoll Disketten
womit die Diskette "formatiert" ,d.h. zum Beschreiben vorbereitet wird.
Bei einer beschriebenen Diskette gehen so alle Daten verloren!

Im folgenden wird angenommen, daß Sie mit der Festplatte D: arbeiten;
mit **"Disk"** ist die **Festplatte** (C:,D:..) oder **Diskette**(A:,B:) gemeint.

Jetzt starten Sie das WATFOR77-System (automatisch mit Editor WEDIT):
WATFOR (und Enter-Taste)
Damit haben Sie automatisch auf Festplatte (Laufwerk D:) umgeschaltet.

Wollen Sie Ihre Programme und Daten auf Ihre eigene Diskette schreiben, schalten Sie um auf Diskettenlaufwerk A: oder B: mit
SYS A: bzw. **SYS B:** (und 2 mal die Enter-Taste)

Sie befinden sich jetzt im Workspace mit dem Namen "no_name" (s. 2. Zeile) des Editors WEDIT, der in das WATFOR77-System integriert ist. Sie haben alle Editor-Kommandos zur Verfügung, wie sie im Skriptum zu WEDIT beschrieben sind. Mit dem Betriebssystem MS/DOS haben Sie nun nichts mehr zu tun. Wie geht es jetzt weiter?

1. Schritt: Workspace bekommt einen Namen (oder Laden File)

Geben Sie Ihrem Programm, welches Sie noch schreiben wollen, einen unverwechselbaren Namen anstelle von "no_name" :
Name *neuername* z.B. Name Prog1
Er kann bis zu 8 Buchstaben oder Ziffern enthalten. Leerzeichen als Teil des Namens sind verboten: z.B. ist Name P1 2 fehlerhaft!

WATFOR stellt hinter P1 automatisch die Filenamen-Erweiterung .FOR

Beispiel: N P1 (das Kommando "Name" ist hier zu "N" abgekürzt) gibt dem Workspace den vollständigen Namen "P1.FOR" . Für die meisten Kommandos genügt P1 allein ohne ".FOR" (außer für ERAse und RENname).

Wenn Sie bereits ein Programm auf Diskette gespeichert haben, geben Sie im Schritt 1 das Kommando Edit mit dem Namen des Diskettenprogramms:
Edit *programmname*
wobei die Erweiterung .FOR wegbleiben kann. Das Kommando Name ist dann überflüssig, da Edit dem Workspace den Programmnamen von Diskette gibt.

2. Schritt: Schreiben / Editieren des Programms

Schreiben Sie neue Zeilen mit der **F5-Taste** (Line Insert).
Eine Zeile kann kopiert werden mit **F6** und **2 mal Shift_F6**,
(Line Delete, Line Undelete), d.h. die Zeile wird erst gelöscht (und gleichzeitig aufbewahrt), durch Drücken von Shift_F6 sogleich wieder eingefügt und steht für weiteres Einfügen an einer bel. Stelle bereit:
Zeile verschieben: **F6**, dann **F9 / F10** Line Up / Down, dann **Shift_F6**.

3. Schritt: Ausführen des Programms

Sie lassen das Programm laufen mit dem Kommando
RUN oder **Alt_F1** (Alt und F1 gleichzeitig)
Sie erhalten alle Meldungen von WATFOR77 am Schirm gelistet; falls kein Error dabei war, wird das Programm automatisch ausgeführt und Sie erhalten die Ergebnisse ebenfalls am Schirm.

Falls Sie die Anweisung READ* verwenden, wird die Eingabe der Daten interaktiv am Schirm erwartet (es gibt aber auch die Möglichkeit, die Daten in einem File auf Disk zu halten, s.u.).
Wenn Sie die Ergebnisse nicht am Schirm haben wollen, sondern auf Diskette (zusammen mit dem Programmlisting), geben Sie das Kommando
RUN/NOXType oder **Alt_F3** (Alt und F3 gleichzeitig)

Wollen Sie die gesamte Bildschirm- Ein- und Ausgabe (PRINT und READ) außer am Schirm auch noch im Listing, geben Sie
RUN/LOgio oder **Alt_F5** (Alt und F5 gleichzeitig)

Die Ausführung des Programms bzw. der Compilation kann mit den Tasten
Strg_Untbr (Strg und Untbr gleichzeitig)
angehalten werden. Dies ist besonders für Endlosschleifen (s.u.) gedacht. Auf die Frage "Do you really want to stop" antworten Sie "y". Wollen Sie nach einem READ* (der Aufforderung zur Eingabe am Schirm, s.u.) das Programm abbrechen: **Strg_Untbr** und dann **Enter** drücken.

4. Schritt: Ergebnisse / Programmlisting

Mit dem Kommando RUN wird automatisch ein File auf Diskette angelegt mit dem Namen des Programms, jedoch Kennzeichnung .LST (statt .FOR). z.B. erzeugt das Programm P1 (also P1.FOR) das Listing-File P1.LST
Wollen Sie sich das Listing ansehen, so geben Sie das Kommando
Edit *programmname*.LST (also z.B. E P1.LST)
womit der Editor einen 2. Workspace anlegt. Die Rückkehr in den 1. Workspace zum Verbessern / Erweitern Ihres Fortranprogramms geht mit
Edit oder **Shift_F7**
womit der 2. Workspace verlassen wird. Dieser bleibt jedoch für ein weiteres Edit erhalten, sodaß Sie mit E (oder Shift_F7) zwischen Listing und Quellprogramm hin und her schalten können. Doch werden i.a. Änderungen des Listings kein lauffähiges Programm ergeben!

Bevor Sie die neue Version Ihres Quellprogramms mit RUN laufen lassen, geben Sie im Workspace des (ungeänderten) Listings das Kommando
QUIT oder **Strg_F6**
womit das Listing gelöscht wird und Sie später mit Edit P1.LST das neue Listing bekommen (sonst geht Edit P1.LST wieder in das alte!).

5. Schritt: Wie 1. oder 2. oder Editor verlassen

Weiter mit Schritt 2 (Bearbeitung desselben Programms) oder mit Schritt 1 (Bearbeitung eines anderen Programms): dabei wird mit
Put *oder* **Put** *neuername*
eine dauerhafte Kopie mit altem (Put allein) oder neuem Namen erstellt.

Am Ende Ihrer Arbeit verlassen Sie den Editor mit
EXit
Dabei wird automatisch die aktuelle Version Ihres Workspaces (das Programm) auf Disk gesichert (mit dem vollen Namen). Wollen Sie das nicht, verlassen Sie den laufenden Workspace unwiderruflich mit
QUIT
Alle vorhandenen Workspaces werden unwiderruflich gelöscht mit
CANcel
Erst nach EXit / QUIT nehmen Sie Ihre Diskette aus dem Laufwerk.

Hilfen für Fortranprogramme am PC

1. Voreinstellung Tabulator-Taste:
springt von Sp. 1 auf Sp. 7, sodann in Zweierschritten zum bequemen Einrücken von Blöcken (WHILE, IF usw.); mit Shift auch rückwärts.

2. Automatischer Piepston in Spalte 72
Spalte 72 ist durch einen Piepston markiert; schreiben Sie statt in Sp. 73 in der nächsten Zeile ab Sp. 7 weiter (also F5 und Tabulator); vergessen Sie dabei nicht, etwas (ungleich 0) in Sp. 6 einzutragen!

3. Großbuchstaben ohne Shift-Taste:
Der Fortran77-Standard fordert, daß alle Anweisungen in Großbuchstaben geschrieben sind. WATFOR erlaubt auch Kleinbuchstaben (entsprechend dem Fortran90-Standard), die von Fortran jedoch nicht von Großbuchstaben unterschieden werden; sie dienen nur der besseren Lesbarkeit.
Das Schreiben von Großbuchstaben erleichtern Sie sich durch einmaliges Drücken der Taste "Groß" (wird durch ein grünes Licht rechts oben angezeigt); das Drücken der Shift-Taste bei Buchstaben entfällt damit; alle Ziffern und Sonderzeichen schreiben sich aber weiterhin wie gewohnt.

Programmierte Tasten mit Alt:

4. Kommando **RUN/NOEX**tensions auf **Alt_F1**
Das Kommando startet ein Fortran-Programm und unterdrückt Extension-Meldungen. Besonderheit: Man kann Alt_F1 auch mitten aus der Text-Area heraus drücken (man muß den Cursor nicht mit F4 nach unten bewegen).

5. Kommando **RUN/NOEX/NOXT**ype auf **Alt_F3**
Wie oben, jedoch werden die Ergebnisse (PRINT*,...) ins Listing-File geschrieben, dh. es ist kein Bildschirm-Dialog möglich.

6. Kommando **RUN/NOEX/LO**gio auf **Alt_F5**
Die gesamte Bildschirm- Ein- und Ausgabe ist am Schirm sichtbar und wird zusätzlich im Listing-File vermerkt.

Ausdrucken eines Files

Drucken eines Files über die Netzwerk-Drucker: Sie drücken
Alt_F2 (Alt und F2 gleichzeitig)
und tippen den Namen des zu druckenden Files, dann die Enter-Taste.

Die Kommandos zum Filehandling (Diskette/Festplatte)

DIrectory	zeigt das Inhaltsverzeichnis der Disk
Get *abc*	**kopiert File** *abc* hinter die Current Line
Put *abc*	**schreibt den lfd. Workspace in das File** *abc*
Name *abc*	**gibt dem Workspace den neuen Namen** *abc*
SHow	zeigt die Namen aller Workspaces (Strg_F5)
Edit *abc*	**erzeugt einen 2. (allg. weiteren) Workspace** *abc*
Edit	geht in den 2. (allg. nächsten) Workspace
DONe	geht in den 2. (allg. nächsten) Workspace und schreibt auf Disk zurück, falls Änderung;
BYE	geht in den 2. (allg. nächsten) Workspace und löscht lfd. Workspace, falls keine Änderung
QUIT	geht in den 2. (allg. nächsten) Workspace und löscht den laufenden Workspace
EXIT	verläßt Workspace mit Zurückschreiben auf Disk
CANcel	verläßt unwiderruflich den Editor
ERAse *abc*	**löscht das File** *abc* von der Disk
REName *abc xyz*	**gibt dem File** *abc* **den neuen Namen** *xyz*

C. Wie werden Fortran-Programme geschrieben?

In Fortran gibt es eine feste Spalteneinteilung:

Zeilenlineal:

```
....;....1....;....2....;....3....;....4....;....5....;....6....;....7..
```

Einteilung einer Fortran-Zeile:

```
nnnnnfaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Dabei bedeutet:

n : Nummernfeld: 1-5 Ziffern
für Formatnummern und Anweisungsnummern

f : Fortsetzungsspalte:
mit bel. Zeichen ungleich 0 gehört die Zeile zur vorigen
Vorschlag: man verwende &

a : Anweisungsfeld:
für Fortran-Anweisungen

zweite Möglichkeit für eine Fortran-Zeile:

```
*.............................................................
```

Dabei bedeutet:

* : * in Sp. 1 (anstelle von * kann auch C stehen)
. : beliebiger Kommentar

Der Inhalt ab Spalte 73 bis Zeilenende einer Fortran-Programmzeile wird ignoriert (kann für Kommentar verwendet werden).

Eine Fortran-Anweisung beginnt immer auf einer neuen Zeile.

Falls eine Zeile nicht reicht, kann die Anweisung mit dem Eintrag in die Fortsetzungsspalte der nächsten Zeile fortgesetzt werden (bis zu 19 mal). Die Spalten 7-72 der Fortsetzungszeile(n) verhalten sich so, als ob sie ab Spalte 73 der Anweisung angefügt wären.

Leerzeichen (blanks) können zur besseren Lesbarkeit beliebig in einer Anweisung verwendet werden. Sie haben für Fortran keine Bedeutung, es sei denn, sie treten innerhalb eines CHARACTER-Strings in Hochkomma auf, z.B. ' ' oder 'Otto Meier' als Bestandteil des Textes.

Vermeiden Sie Leerzeichen innerhalb von Namen und Schlüsselwörtern!
Beispiel: INTEGER, niemals INT EGER schreiben.

Innerhalb von Namen kann mit _ gegliedert werden:

Beispiel: Guthaben_alt, Guthaben_neu

Kapitel 1: Mensch - Computer - Programm

Fragen

1. Was kann eigentlich genaugenommen ein Computer?
2. Wenn das alles ist, warum verwenden wir Computer überhaupt?
3. Welche Schritte sind nötig, um ein Problem zu lösen?
4. Was passiert, wenn man einen Fehler macht?

Was kann der Computer?

Wie viele elektronische Bestandteile ein Computer auch haben mag, sie haben alle im wesentlichen die folgenden 3 grundlegenden Eigenschaften:

1. Jeder Computer hat Schaltkreise für arithmetische Operationen, meist + - * / (Addition, Subtraktion, Multiplikation, Division).
2. Jeder Computer hat Schaltkreise, um Entscheidungen zu treffen. Die Entscheidungen sind nicht von der Art: "Wer würde den Krieg gewinnen zwischen Vietnam und den USA?", sondern der Computer kann nur entscheiden, ob eine Zahl kleiner, gleich oder größer ist als eine andere Zahl.
3. Jeder Computer hat die Möglichkeit, mit dem Menschen zu kommunizieren, sonst hätte alle Rechnerei ja keinen Sinn. Die Eingabe von Information geschieht heute meist über Tastatur / Bildschirm. Zur Ausgabe der Information aus dem Computer wird entweder ein Drucker oder ebenfalls der Bildschirm verwendet.

Da alle Computer nur diese 3 Arten von Fähigkeiten haben - einfache Arithmetik, einfache Entscheidungen, Ein- und Ausgabe - , stellt sich die Frage:

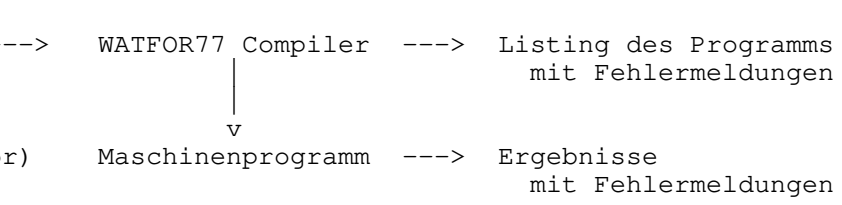
Warum benutzen wir überhaupt Computer?

Es gibt darauf im wesentlichen 3 Antworten:

1. Schnelligkeit.
   Ein heutiger Rechner kann etwa 100 Millionen Additionen pro Sekunde ausführen. Ein Mensch braucht dazu 125 Jahre, wenn er 40 Stunden die Woche arbeitet. Entscheidungen laufen ebenso schnell ab. Das Drucken erfolgt z.T. mechanisch und ist daher wesentlich langsamer, ein Laser-Schnelldrucker am URZ druckt 4000 Zeilen pro Minute.
2. Genauigkeit und Verläßlichkeit.
   Die Berechnungen der Computer sind meist (Ausnahmen s. später) sehr genau und jederzeit reproduzierbar; hin und wieder bricht ein Computer auch zusammen, meistens wegen falscher Bedienung oder fehlerhafter Programmierung, seltener wegen Hardware-Fehlern; meist kann er schnell wieder zum Laufen gebracht werden.
3. Ein großes Problem = eine Menge kleiner Probleme
   Der wichtigste Grund, der für Computer spricht, ist die Möglichkeit, große Probleme mit dem Computer dadurch zu lösen, daß man eine große Zahl kleiner Probleme daraus macht und diese eins nach dem anderen - der Reihe nach - löst; z.B. ist die Erzeugung einer Aufstellung aller Lohnzahlungen in einer großen Firma ein großes Problem. Wir müssen dazu folgendes tun:
   a) Eingabe von: Lohn, Steuer, Pensions-Ansprüche, geleistete Arbeitsstunden usw. für jeden Beschäftigtrn.
   b) Ausführung einiger einfacher arithmetischer Operationen.
   c) Ausgabe einiger Zeilen als Ergebnis der Berechnungen.
   Durch einfaches Wiederholen für jeden einzelnen Fall können wir die Aufgabe lösen.

5 Schritte zur Lösung eines Problems mit dem Computer

1. Beschreibe das Problem
   Ehe ein Problem nicht genau (auf Deutsch) beschrieben ist, hat es wenig Sinn, mit dem Computer zu arbeiten. In der Datenverarbeitung bekommen diejenigen die höchsten Löhne, die die Frage beantworten: "Was, genau, ist es, wozu wir den Computer einsetzen wollen?".
2. Entwickle einen Algorithmus
   Ein Algorithmus ist eine Folge von Anweisungen, wie der Weg zur Lösung des Problems aussehen soll. Es gibt verschiedenen Formen der Darstellung von Algorithmen. Eine dieser Formen wird unten an Beispielen ausführlich dargestellt. Wenn der Algorithmus einmal formuliert ist, ist die Programmierung bedeutend einfacher.
3. Übersetze den Algorithmus in eine Programmiersprache (Fortran).
   Die Anweisungen in der neuen Sprache nennt man Computerprogramm. Es gibt Hunderte solcher Sprachen, genaugenommen versteht aber der Computer nur e i n e Sprache, und diese besteht nur aus Nullen und Einsen ("Maschinensprache"). Das Programm, welches Fortran-Anweisungen in Maschinensprache übersetzt, nennt man Compiler: es ist selbst in Maschinensprache geschrieben.
4. Schreibe das Programm am Bildschirm
   Nachdem das Programm in Fortran geschrieben ist, gibt man es am Schirm in den Computer ein. Das eingetippte Fortran-Programm nennt man Quellprogramm.
5. Lasse das Programm laufen (Compilation, sodann Ausführung)
   Das Laufenlassen ist ein Prozeß mit 2 Schritten:
   a) Das Fortran-Quellprogramm wird vom Compiler in Maschinensprache übersetzt. Gleichzeitig zeigt der Compiler das eingegebene Programm an, zusammen mit Meldungen über die Compilation und über eventuelle Fehler (Verstöße gegen die Fortran-Regeln). Man sagt, diese Vorgänge laufen zur "compile time" ab.
   b) Der 2. Schritt besteht darin, daß der Computer die Anweisungen ausführt, die im Programm enthalten sind. Dies geschieht zur "execution time". Alle Fehler, wie Division durch 0, beenden die Ausführung des Programms, und eine Fehlermeldung wird ausgegeben. Schematisch sieht das so aus:



Beispiel 1.1

Aufgabenstellung: Die Botschaft "Mein erstes Fortran-Programm" soll auf dem Bildschirm erscheinen.

```
      PROGRAM P11
* Beispiel 1.1: Zeige am Schirm: "Mein erstes Fortran-Programm"
      PRINT*, 'Mein erstes Fortran-Programm'
      END
```

Erklärungen:
1. Die 1. Zeile ist der Programm-Kopf. Hier wird der Name des Programms angegeben, hier: P11
2. Die 2. Zeile ist Kommentar. Kommentar beginnt mit * in Sp. 1 und umfaßt die ganze Zeile.
   Kommentare haben auf die Logik des Programms keinen Einfluß.
3. Die übrigen Zeilen enthalten die Anweisungen, die der Computer ausführen soll, eine PRINT-Anweisung und die END-Anweisung.

Beispiel 1.2

Aufgabenstellung: Es soll der Zinsertrag berechnet werden für 100 DM
in 1 Jahr bei einem Zinssatz von 12 %.

```
      PROGRAM P12
* Beispiel 1.2: Zinsertrag für 100 DM in 1 Jahr bei 12%
      REAL Kapital, Zinsertrag
      Kapital    = 100
      Zinsertrag = Kapital * 0.12
      PRINT*,'Kapital:',Kapital,' Zinsertrag bei 12% Zins:',Zinsertrag
      END
```

Erklärungen:
1. Kapital und Zinsertrag sind Variable vom Typ REAL.
   Variable können die Ergebnisse von Berechnungen des Computers enthalten. Alle Variablen werden am Anfang mit ihrem Typ deklariert (s.a. Kap. 2). Der Typ REAL steht für Zahlen mit Dezimalpunkt.
2. Jede Anweisungen beginnt auf einer neuen Zeile. Sie kann auch über mehrere Zeilen gehen, z.B. hätten wir schreiben können:

```
      PRINT*,  'Kapital:',Kapital,
     &         ' Zinsertrag bei 12% Zins:',Zinsertrag
```

   mit einem & in Sp. 6 der Fortsetzungszeile.
3. Die Anweisung "Kapital = 100"
   bedeutet: es wird der Wert 100 genommen und dem Kapital zugewiesen
   `Zinsertrag = Kapital * 0.12`
   bedeutet: es wird der Wert von Kapital mit 0.12 multipliziert; der so errechnete Wert wird der Variablen Zinsertrag zugewiesen.
   `Variable = ...` heißt also:
   der Variablen wird der (berechnete) Wert zugewiesen

Wenn Sie das Programm mit RUN starten, erhalten Sie die Meldung
EXT CC-04 line 3, character encountered is not FORTRAN77 standard
Diese Meldung können Sie ignorieren (sie wird durch die Kleinbuchstaben von "Kapital" und "Zins" verursacht). Starten Sie statt mit "RUN" mit "RUN/NOEX" oder besser mit Alt_F1 (programmierte Taste, s.o. Kap. 0), und die Meldung wird unterdrückt.

Beispiel 1.3

Aufgabenstellung: Wie 1.2, aber Zinsertrag für 10 Jahre berechnen

```
      PROGRAM P13
* Beispiel 1.3: Wie 1.2, aber Zinsertrag für 10 Jahre berechnen
      REAL Kapital, Zinsertrag
      INTEGER Jahr
      Kapital = 100
      Jahr    = 1991
      WHILE (Jahr .LE. 2000)
         Zinsertrag = Kapital * 0.12
         PRINT*,      Kapital,Zinsertrag
         Kapital    = Kapital + Zinsertrag
         Jahr       = Jahr + 1
      ENDWHILE
      END
```

Erklärungen:
1. Die WHILE-Anweisung führt den Block der Anweisungen bis ENDWHILE solange aus, wie Jahr kleiner oder gleich 2000 ist: "Jahr .LE. 2000"
   Dabei steht .LE. für "Lower or Equal" (kleiner oder gleich).
   Eine Wiederholung von Anweisungen nennt man eine Schleife.
   Innerhalb der Schleife WHILE ... ENDWHILE wird das Jahr um 1 erhöht, sodaß nach 10 Schritten 2000 erreicht ist. Beim 11. Mal ist die Bedingung, daß das Jahr kleiner oder gleich 2000 ist, nicht mehr erfüllt; das Innere der Schleife wird nicht mehr ausgeführt.

2. Es gibt 6 Vergleichsmöglichkeiten (Vergleichs-Operatoren):
   .LT. kleiner .GT. größer .NE. ungleich
   .EQ. gleich .LE. kleiner oder gleich .GE. größer oder gleich
3. Die Anweisung "Jahr = Jahr + 1" bedeutet: Es wird der Wert von Jahr + 1 berechnet und das Ergebnis der Variablen Jahr zugewiesen.

Übungen

1. Ändern Sie Beispiel 1.3 für einen Zinssatz von 9 %.
2. Ändern Sie Beispiel 1.3 für 1000 DM und 5 Jahre.
3. Ändern Sie Beispiel 1.3, sodaß Jahr, Kapital, Jahresertrag und der gesamte Zinsertrag angezeigt werden.

Was passiert, wenn man einen Fehler macht?

Wenn eine Fortran-Anweisung Fehler enthält (Syntax-Fehler), wird nach dem RUN die entspr. Fehlermeldung angezeigt mit Angabe der Zeile, wo der Fehler auftrat ("line ..."). Drücken Sie Enter und tippen Sie die Zeilennummer (n) als Kommando ein:
Tippen Sie :n und Enter):
die fehlerhafte Zeile erscheint hell (Current Line).
Im folgenden werden einige häufig auftretende Fehler und deren Behebung besprochen. Zunächst einige **Syntax-Fehler** (compile time error):

Beispiel 1.4

```
      PROGRAM P14
* Beispiel 1.4: 5 verschiedene Syntax-Fehler
      REAL x,y,z
      x = 2,0
      y = x*x - 3x
      z = (x+y)/y+2,0)
      PRINT*x,y,z
      END
```

Nach dem Programmstart (Alt_F1) erscheinen folgende **Syntax-Fehler**:

```
  *ERR* SX-02 line 4, at end of statement, bad sequence of operators
  *ERR* SX-16 line 5, column 18, missing operator
  *ERR* PC-04 line 6, column 23, unmatched parentheses
  *ERR* SX-10 line 7, column 12, missing comma
```

Geben Sie das Kommando "4" für line 4 usw.

line 4: hier ist nicht sofort zu entnehmen, daß anstelle des Kommas in einer Dezimalzahl der Dezimalpunkt stehen muß.
Korrektur: schreiben Sie statt , einen .

line 5: hier ist eine Spalte (column 18) angegeben: drücken Sie F4 und bewegen Sie den Cursor auf Spalte 18 (sichtbar in der 2. Zeile rechts oben, wo hinter der 4 (=Zeile 4) die Spalte steht)
Korrektur: 3 mal X schreibt man in Fortran "3*X".

line 6: hier fehlt eine Klammer; die Spaltenangabe ist irreführend, da der Compiler das Unglück erst am Zeilenende merkt!
Korrektur: die Klammer-auf hinter / fehlt.

line 7: hier fehlt das Komma hinter dem *
Korrektur: fügen Sie hinter dem * ein , ein

Haben Sie alles verbessert und drücken wieder Alt_F1, so tritt ein neuer, bisher unbemerkter Fehler auf:

```
  *ERR* SX-16 line 6, column 23, missing operator
```

Der Fehler ist wieder, daß ein , statt . (Dezimalpunkt) dasteht.
Wir lernen daraus: Obwohl beidesmal derselbe Fehler gemacht wurde, nämlich eine REAL-Zahl 2,0 zu schreiben statt 2.0, meldet der Compiler verschiedene Fehler mit unterschiedlicher "Erklärung": oben war es SX-02 "bad sequence of operators" und jetzt SX-16 "missing operator".

Execution-Time-Fehler (z.B. Division durch 0) geben die Nummer der Fortran-Anweisung an ("statement ..."). Die Nummer zeigt aber nur dann auf die fehlerhafte Zeile, wenn weder Kommentar noch Fortsetzungszeilen vorhanden sind (was meist nicht so ist). Zählen Sie also nur die Fortran-Anweisungen, oder noch besser, editieren Sie das Listing-File und schauen Sie nach, welche Zeile das "statement ..." war (im Listing-File in Sp. 7 links vor jeder Fortran-Zeile angegeben).
Zur Erinnerung: zum Programm z.B. P14.FOR gehört das Listing P14.LST

Beispiel 1.5

```
      PROGRAM P15
* Beispiel 1.5: Division durch 0
* identisch mit dem (inzwischen hoffentlich) verbesserten Beispiel 1.4
      REAL x,y,z
      x = 2.0
      y = x*x-3*x
      z = (x+y)/(y+2.0)
      PRINT*,x,y,z
      END
```

Resultat:
```
*ERR* KO-01 floating point divide by zero
TraceBack: Executing in P15, statement 5
```
Erklärung:
Der Fehler, eine Division durch 0, tritt auf, weil Y den Wert -2 in Zeile 6 bekommt und in Zeile 7 durch Y+2.0 also durch 0 dividiert wird. Das Traceback nennt "statement 5". Dies muß nicht die 5. Zeile sein. Im Beispiel ist es Zeile 7. Sehen Sie im Listing nach statement 5. Dort sind die Anweisungen (statements) links vorn durchlaufend numeriert.

Beispiel 1.6

```
      PROGRAM P16
* Beispiel 1.6: Rechnen mit undefiniertem Wert
      REAL a,b
      b = a*a
      a = 7.0
      PRINT*,b
      END
```

Resultat:
```
*ERR* UV-05 the value of A is undefined
TraceBack: Executing in P16, statement 3
```
Erklärung:
Der Fehler besteht darin, daß bei Ausführung der Anweisung "B = A*A" die Variable A noch keinen Wert besitzt (dies wird erst in der nächsten Anweisung "nachgeholt").

Beispiel 1.7

```
      PROGRAM P17
* Beispiel 1.7: Ausgabe eines undefinierten Wertes
      REAL b
      PRINT*,b
      END
```
Resultat: ????????

Erklärung: Bei der Ausführung der PRINT-Anweisung hatte B noch keinen Wert. Die Ausgabe eines undefinierten Wertes gilt aber nicht als Fehler, sondern wird mit ???????? angezeigt.

Beispiel 1.8

Aufgabenstellung: Berechnung der Summe der ersten 50 natürlichen Zahlen und deren Durchschnittswert

```
      PROGRAM P18
* Beispiel 1.8: Summe der ersten 50 natürlichen Zahlen mit Durchschnitt
      INTEGER Summe, Zahl, Durchschnitt
      Summe = 0
      Zahl  = 1
      WHILE( Zahl .LE. 50 )
         Summe = Summe + Zahl
         Zahl  = Zahl + 1
      ENDWHILE
      Durchschnitt = Summe / 50.0
      PRINT*, 'Durchschnitt =', Durchschnitt
      END
```

Beispiel 1.9

Schritt 1: Idee zur iterativen Berechnung der Quadratwurzel:
Ausgangspunkt: Gegeben sei die Zahl A.

1. Näherung: $X_1 = A/2$

2. Näherung: $X_2 = \frac{1}{2} (X_1 + A/X_1)$

3. Näherung: $X_3 = \frac{1}{2} (X_2 + A/X_2)$ usw.

Schritt 2: Algorithmus:

1) setze Zahl = 726.4
2) setze alte Näherung = Zahl (der Wert wird unten in 5) benutzt; ansonsten ist er beliebig; darf aber nicht gleich Zahl/2 sein. Warum?)
3) setze Näherung = Zahl/2
4) setze Zähler = 1
5) WHILE Differenz alte und neue Näherung größer als 0.001
   5.1) erhöhe Zähler um 1
   5.2) setze alte Näherung = Näherung
   5.3) setze Näherung = ... s.o. Formel
6) Schreibe den Wert der Zahl, der Näherung und des Zählers

Schritt 3: Übersetzung in Fortran:

```
      PROGRAM P19
* Beispiel 1.9: Iterative Berechnung der Wurzel aus 726.4
*          Stop, wenn die letzte Verbesserung kleiner als 0.001 ist
      REAL Zahl, alte_Naeherung, Naeherung
      INTEGER Zaehler
      Zahl           = 726.4
      alte_Naeherung = Zahl
      Naeherung      = Zahl / 2.0
      Zaehler        = 1
      WHILE( ABS (alte_Naeherung - Naeherung) .GT. 0.001)
        alte_Naeherung = Naeherung
        Naeherung      = ( alte_Naeherung + Zahl/alte_Naeherung ) / 2.0
        Zaehler =      Zaehler + 1
      ENDWHILE
```

```
      PRINT*,'Wurzel aus',Zahl,' nach',Zaehler-1,' Iterationen ist',
     &      Naeherung)
      END
```

Anm.: Die Fortran-Funktion ABS berechnet den Absolutbetrag, s.u. Kap. 2

Übungen

Modifizieren Sie das Beispiel 1.9 so, daß

1. a) jede Näherung ausgegeben wird
   b) Stop nach 6 Iterationen
   c) Stop, wenn die Änderung der Näherung kleiner als 0.1 % des Wertes der Zahl ist. Ist das ein guter Gedanke?
2. Mit der Fortran-Funktion SQRT kann der (hoffentlich!) genaue Wert der Quadratwurzel bestimmt werden. Man modifiziere das Programm so, daß Wurzel aus 7 auf 0.001 genau berechnet wird und vergleiche mit dem Wert von SQRT(Zahl).
3. Was passiert, wenn das Programm die Wurzel aus einer negativen Zahl berechnen soll? (ausprobieren!) Was wäre zu tun, um auch diesen Fall vom Programm richtig zu verarbeiten?

Programmierstil

Gut geschriebene Programme haben folgende Eigenschaften:

1. Viele Kommentare
2. Leerzeichen trennen einzelne Symbole, Operatoren und Programmteile
3. Namen der Variablen nach ihre Bedeutung
4. Anweisungen einrücken zwischen WHILE und ENDWHILE

Zusammenfassung

A. Stichworte:

Bit, Byte
Hardware, Software
Maschinensprache, Hochsprache, Fortran
Fortran-Programm, Anweisung, Zuweisung, Kommentar
Compiler, Compilation, Compile Time
Execution, Execution Time
Extension, Warning, Error
Syntax-Fehler, Compile Time Error
Fehler bei der Ausführung, Execution Time Error
Character, Zeichen, Zahl, Ziffer, Buchstabe
ganze Zahl, gebrochene Zahl, Dezimalzahl, Dezimalpunkt
Algorithmus, iterativer Algorithmus

Machen Sie sich für diese Stichworte deren Bedeutung und den Zusammenhang klar. Fragen Sie, wenn Ihnen etwas unklar ist.

B. 5 Schritte der Programmentwicklung:

1. Definieren Sie das Problem.
2. Entwickeln Sie einen Algorithmus zur Lösung des Problems.
3. Übersetzen Sie diesen in eine Computersprache (Fortran).
4. Geben Sie die Anweisungen in den Computer (am Bildschirm).
5. Lassen Sie den Computer die Anweisungen ausführen.

C. Fortran-Typen

Fortran verarbeitet 3 Arten von Zahlen:
ganze Zahlen: **Typ INTEGER**
gebrochene Zahlen: **Typ REAL**
komplexe Zahlen: **Typ COMPLEX** , wird im Skript nur am Rande erwähnt

Weiter gibt es noch in Fortran:

logische Werte	.TRUE. und .FALSE. als wahr / falsch) s. Kap. 3
	der **Typ LOGICAL** wird am Ende von Kap. 5 behandelt
bel. Zeichen	Text in Hochkomma: '...'
	der **Typ CHARACTER** wird in Kap. 8 behandelt.

D. Fortran-Programme haben die Form

```
PROGRAM name
Deklarations-Teil (Angabe von Typ und entspr. Variablen)
Anweisungs-Teil (executable statements)
END

Kommentar (* in Sp. 1) kann an beliebiger Stelle stehen.
```

E. In den Beispielen wurden folgende Anweisungen benutzt:

1. PROGRAM ... gibt dem Programm einen Fortran-Namen
2. Kommentarzeilen mit * in Spalte 1
3. INTEGER ... | REAL ... vereinbaren Variable als INTEGER | REAL
4. Die Zuweisung ... = ... weist der links stehenden Variablen einen Wert zu: Variable = Wert
5. Die WHILE-Anweisung steuert, wie oft eine Schleife, d.h. die Anweisungen bis ENDWHILE, ausgeführt werden.
6. Die PRINT-Anweisung zeigt Texte und Ergebnisse am Schirm.
7. Die eingebauten Funktionen ABS für den Absolutbetrag und SQRT für die Quadratwurzel.
8. END ist die letzte Zeile des Programms, beendet den Programmlauf.

F. Fehlermeldungen.

Der Compiler unterscheidet 3 Stufen von Meldungen:

1. Extension

Erweiterung (Vorgriff auf den Fortran90-Standard).
Das Anzeigen dieser Meldungen kann durch die Verwendung der programmierten Taste Alt_F1 (statt des Kommandos RUN) unterdrückt werden (wirkt wie RUN/NOEX).

2. Warning

Stellt keinen eigentlichen Fehler dar, sollte jedoch ernst genommen werden (z.B. der Hinweis auf Anweisungen, die niemals ausgeführt werden können u.ä.).

3. Error

a) Syntax-Fehler: Fehler, die zur sog. "compile time" auftreten. Diese Fehler werden bis Programmende gesammelt; falls vorhanden, wird das Programm nicht zur Ausführung gebracht.

b) Fehler während der Ausführung (execution time error). Beim ersten Auftreten wird der Programmlauf gestoppt. Mit Hilfe des Debuggers kann man an dieser Stelle nach einer lokalen Korrektur evtl. den Programmlauf fortsetzen.

Die Korrektur von Fortran-**Syntax-Fehler**n ist im Prinzip einfach, doch muß man beachten, daß der Compiler für denselben Schreibfehler in verschiedenen Zeilen u.U. verschiedene Fehlermeldungen erzeugt, s.o. Beispiel 1.4; umgekehrt kann es sein, daß für einen einzigen Fehler mehrere Fehlermeldungen ausgegeben werden, von denen vielleicht nur eine den Fehler zutreffend beschreibt. Beheben Sie also die Fehler unbedingt der Reihe nach; im Zweifel nur einen Fehler auf ein Mal (sodann RUN); der Compiler ist so schnell, daß Sie dadurch kaum Zeit verlieren!
Execution time error sind meist sehr gefürchtet, da viele Fortran-Compiler einen im Stich lassen mit sinnvollen Meldungen und allenfalls kryptische Systemmeldungen erzeugen, womit man nur wenig anfangen kann. Anders der WATFOR-Compiler: Er besitzt eine ausgezeichnete Execution Time Fehlerdiagnostik.
Kommt trotz allem das gewünschte Ergebnis nicht heraus, kann evtl. noch der Debugger helfen, s. Anhang C.

Aufgaben

1. Berechnen Sie die Summe von 127 und 584

2. Schreiben Sie Ihren Namen in großen Buchstaben wie unten; benutzen Sie eine Folge von PRINT-Anweisungen.

```
*****  *   *  *   *   ***
*      ** **  ** **  *   *
*      * * *  * * *  *   *
*****  *   *  *   *  *****
*      *   *  *   *  *   *
*****  *   *  *   *  *   *
```

3. Schreiben Sie ein Programm, welches die Quadrate und Kuben der Zahlen von 1 bis 20 zeigt.

4. Modifizieren Sie das Programm, daß es Quadrate und Kuben der ungeraden Zahlen von 1 bis 20 zeigt.

5. Sie haben einen Job angeboten bekommen, wo Sie am 1. Tag nur 1 Pfennig verdienen, am nächsten aber das doppelte usw. Wieviel bekommen Sie insgesamt in 30 Tagen? In Mark und Pfennig?

6. Schreiben Sie ein Programm, welches zeigt, wieviel Geld Sie am Jahresende schulden, wenn Sie 1000 DM zu 10 % borgen und 200 DM am Jahresende zurückzahlen. Das Programm soll Jahr, Schuld und Zinsen an jedem Jahresende zeigen, solange, bis die Schuld beglichen ist.

7. Tabellieren Sie (berechnen und ausdrucken) die Werte von $y= (x^2-x+2) / (x + 3)$ für x=0,1,...,10

8. Tabellieren Sie (berechnen und ausdrucken) die Werte von $y=3x^2-2x+7$ für x=1,3,5,...,21

9. Berechnen Sie die Summen von
   0^3, $0^3+ 1^3$, $0^3+ 1^3+ 2^3$, ... , $0^3+ 1^3+ 2^3+ ...9^3$.
   Benutzen Sie den Operator ** für "hoch" (X hoch 3: X**3).
   Drucken Sie die Quadratwurzel jeder Summe aus (Funktion SQRT).
   Welches allgemein gültiges Ergebnis ergibt sich?

10. Die sog. Fibonacci-Zahlen sind 0,1,1,2,3,5,8,..., wobei jede Zahl die Summe der beiden vorhergehenden ist. Berechnen Sie die ersten 15 Fibonacci-Zahlen.

11. Benutzen Sie den Operator ** (X hoch 2: X**2) zur Berechnung von
    x, x^2, x^4, x^8, x^{16} für x=1.25. Verwenden Sie einen Algorithmus mit Schleife.

12. Berechnen Sie für x=1.25 die ersten 8 Glieder der Folge
    $x, x^{\frac{3}{2}}, x^{\frac{6}{2}}, x^{\frac{10}{3}}$ ... (Man beachte, daß Exponent und Nenner sich jeweils um 2,3,4,... vergrößern)

Kapitel 2: Zahlen, Arithmetik und Variablen

Fragen

1. Wie werden INTEGER-Zahlen und REAL-Zahlen definiert und benutzt?
2. Wie sind die Regeln für die Berechnung arithmetischer Ausdrücke?
3. Wie werden Variable definiert?
4. Wie sehen die beiden Methoden aus, Variablen einen Wert zu geben?

Ganze Zahlen (INTEGER)

INTEGERs bestehen aus Ziffern ohne Dezimalpunkt, evtl. mit vorangestelltem Vorzeichen, z.B. 123 -4375 +2 0
Der Zahlenbereich für INTEGER-Zahlen ist 2147483647 ... -2147483648;
oder -2hoch31, -2hoch31 +1,...,-1,0,1,..., 2hoch31 -1

Mit INTEGERs können folgende Operationen ausgeführt werden:

- `+` Addition
- `-` Subtraktion
- `*` Multiplikation
- `/` ganzzahlige Division zweier INTEGERs (Ergebnis INTEGER) der Divisionsrest geht verloren
- `**` Exponentiation

Es gelten die üblichen Rechenregeln, wobei aber bei / ganzzahlig der gebrochene Teil des Ergebnisses verschwindet (ohne Rundung).

Prioritäten bei der Auswertung:

1. Funktionen
2. Klammern (innerste zuerst)
3. ** von rechts nach links
4. * / von links nach rechts
5. + - von links nach rechts

Beispiele für das Rechnen mit INTEGERs:

```
3+2*4          ->   11
5+2**3*7       ->   61
-4**2          ->  -16
4**2**3        ->   65536
5/2            ->   2
5/-2           ->   *ERR* - richtig ist 5/(-2)  ->  -2
3*5/4*3        ->   9
1234**50       ->   0  !!  der gültige Zahlenbereich
                       von ca. 2 Mrd. wird überschritten
```

Übungen

1. Schreiben Sie den folgenden Ausdruck um, indem Sie so viele Klammern wie möglich entfernen, ohne den Wert zu ändern:
   ((-(5*2)**3)+(14/(2**(-3))))
2. Schreiben Sie einen INTEGER-Ausdruck mit dem Wert 20
   a) mit 1 Multiplikation und 1 Subtraktion
   b) 3 Divisionen
   c) 1 Exponentiation, 1 Division und 1 Addition
   d) 2 Klammerpaaren (die auch notwendig sind)
   e) 2 Exponentiationen und 1 Division

Zahlen mit Dezimalpunkt (REAL)

REAL-Zahlen bestehen in Fortran aus 3 Teilen:

1. dem ganzzahligen Teil
2. dem gebrochenen Teil (wahlweise)
3. dem Exponent (wahlweise)

REAL-Zahlen bestehen aus dem gebrochenen Teil der REAL-Zahl, z.B. 7.375, und wahlweise einem Exponenten E und einer ganzen Zahl mit Vorzeichen. Doppeltgenaue REAL-Zahlen müssen den Exponenten D mit sich führen.

Beispiele z.T. mit Fehlern (***):

2,457.21	Komma nicht erlaubt ***
-7.	gebrochener Teil der REAL-Zahl kann wegbleiben
.735	ganzzahliger Teil der REAL-Zahl kann wegbleiben
92.4E3.8	Exponent hinter E muß ganzzahlig sein ***

im Programmm:	ergibt mit PRINT*, PC	mit PRINT*, Workstation
7.375	7.3750000	7.375000000
-0.00621E-3	-6.2100000E-006	-0.6209999810E-05
897524.0E27	8.9752400E+032	0.8975240144E+33
0.0	0.0000000	0.0000000000E+00
1.23456789012345	1.2345680	1.234567881
1.23456789012345D0	1.234567890123450	1.23456789012345003

Der Exponentenbereich für REAL-Zahlen ist für PC und Workstation -38 ... +38 (REAL einfach genau) bzw. -308 ... +308 doppelt genau. Dazu kommen ca. 7 bzw. 15 Dezimalen (ganzzahliger und gebrochener Teil). Dies entspricht dem sog. IEEE-Standard.
Die 5 REAL-Grundrechenarten haben dieselbe Priorität wie bei INTEGER.
Die Division wird mit demselben Zeichen gechrieben wie bei INTEGER:
/ : REAL-Division mit mind. 1 REAL, evtl. 1 INTEGER (Ergebnis REAL)

Beispiele:

im Fortranprogramm:	PRINT*, am PC	PRINT*, an der Workstation
10.0 / 5.0 + 3.56	5.56	5.559999943
6.25-3.0**2.0	-2.75	-2.750000000
-30.0/8.0/100.0	-0.0375	-0.3750000149E-01
-2.0**6.0	-64.	-64.00000000
2.25/15.0*(-0.6)	-0.9	-0.9000000358E-01
3.0*(3.0**2.0/8.0-1.0)	0.375	0.3750000000
2**32	0 !!	0
2**31	-2147483648	-2147483648
2.0**31	2.1474840E+009	0.2147483648E+10
2D0**32	4294967296000000D+009	4294967296.00000000
(-2.0)**3	-8.0000000	-8.000000000
(-2.0)**3.0	*ERR*	-8.000000000

Am PC versucht es der Compiler über Logarithmen, was mit -2. nicht geht.

Werden die arithmetischen Operatoren (+,-,*,/) auf eine REAL- und eine INTEGER-Zahl angewandt, wird die Integer-Zahl vorher in REAL umgewandelt - "gemischte Arithmetik": vergl. mit oben:.
10.0 / 5 + 3.56 -> 5.560000 (PC; 5.559999943 an der Workstation)
2. Zahl ist INTEGER, gerechnet wird also 10.0 / 5.0

Regel: Vermeiden Sie wo möglich gemischte Arithmetik:
benutzen Sie in 1 Anweisung durchweg INTEGER oder durchweg REAL
Ausnahme: 2.0**3 : ganzzahliger Exponent immer als INTEGER schreiben!

Konstante, Variable

Konstante sind Zahlen, z.B. "0.1" (auch Strings, s. später), die während der Ausführung des Programms gleich bleiben. Variable sind Größen, die unterschiedliche Werte annehmen können.
Die Benutzung einer Variablen geht in 3 Schritten:
1. Auswahl eines gültigen Namens
2. Mitteilung an den Compiler, um welchen Typ es sich handeln soll (z. B. INTEGER, REAL, CHARACTER usw.) "Typ-Deklaration"
3. Zuweisung eines Wertes, bevor ein Wert entnommen wird. Dies ist grundsätzlich auf 2 Arten möglich:
   a) Zuweisungsanweisung (Anweisung der Form Variable = Ausdruck)
   b) Einlesen eines Wertes mit der READ-Anweisung, s.u.

```
        Fortran-Namen:
 1. Maximale Länge: 6 Buchstaben oder Ziffern (Fortran77)
    Die meisten Compiler gestatten längere Namen (31 Zeichen)
 2. Erstes Zeichen ein Buchstabe
 3. weiter mit Buchstaben oder Ziffern, keine Sonderzeichen
 4. Blanks sollen innerhalb eines Namens nicht verwendet werden
```

Beispiele:	**gültig**	**zu vermeiden**	**mehr als 6 Zeichen**	**ungültige Namen:**
	MUTTER			9D
	REAL	*		A(I
	RR2			RR#2
	NETPAY		Mueller	NET-PAY
	A9A9		Net_Pay	

Regel: vermeiden Sie Namen (s.o. *), die schon eine andere Bedeutung in Fortran haben, also z.B. INTEGER, REAL, WHILE usw.!

Anmerkung: Die meisten Compiler akzeptieren auch Kleinbuchstaben und das Unterstreichungszeichen (wie im Fortran90-Standard).

Erklärung des Typs (Deklaration)

1. Art: Voreinstellung der Typen: ohne Deklaration nimmt Fortran folgende Typen an ("Erste-Buchstaben-Regel"):
   Ist der 1. Buchstabe I,J,K,L,M,N : INTEGER; sonst REAL
2. Art: explizite Deklaration des Typs:

```
INTEGER A,Ops,Prt,ZZ,Sum
REAL N,J96,I,BAF
REAL*8 D                  REAL mit doppelter Genauigkeit
```

Hinter INTEGER und REAL usw. kein Komma!
Eine Variable kann nicht 2 Typen, z.B. REAL und INTEGER, besitzen.
NB: Ausschalten der "Erste-Buchstaben-Regel" erfolgt mit der Anweisung

```
IMPLICIT NONE
```

unmittelbar hinter der PROGRAM-Anweisung. Empfehlenswert zur Vermeidung von Schreibfehlern, da sonst automatisch eine neue Variable angelegt wird!

Zuweisung von Werten an Variable

Die allgemeine Form der Zuweisung ist **variable = wert**

Bedeutung: der rechts vom = stehende Wert wird berechnet und der Variablen auf der linken Seite zugewiesen.

Beispiele:

```
A      = 3.0 * 8.0 / 2.0**4
JJ     = MOM * (3**I + 1)
A      = 10
L23456 = 3.7
```

Dabei sei angenommen, daß die Variablen MOM und I bereits Werte enthalten, ansonsten kann die Anweisung nicht ausgeführt werden.
Im Fall A=10 wird A die INTEGER-Konstante 10 zugewiesen; in der Zeile darunter umgekehrt einer INTEGER-Variablen eine REAL-Konstante.
In beiden Fällen wird automatisch eine Typumwandlung vorgenommen.
A erhält den Wert 10.0, L23456 den Wert 3 (.7 fällt weg!).

Einlesen von Werten

Das Einlesen eines Wertes gibt einer Variablen denselben Wert wie die Zuweisung dieses Wertes im Programm. Der Wert ist jetzt aber nicht mehr fester Bestandteil des Programms, sondern kann am Bildschirm eingetippt oder vorher in ein Datenfile geschrieben werden.

Beispiel 2.1

```
       PROGRAM P21
* Beispiel 2.1: Berechnung der Fakultät von n
       INTEGER FakN, Mult, n
       FakN = 1
       Mult = n
       WHILE (Mult .GE.1)
         FakN = FakN * Mult
         Mult = Mult - 1
       ENDWHILE
       PRINT*,n,' Fakultät =',FakN
       END
```

Damit erhalten Sie die Meldung *ERR* UV-05 the value of N is undefined
Setzen Sie hinter der Zeile "INTEGER ..." die zwei Anweisungen ein:

```
      PRINT*,'bitte Wert fuer n eingeben'
      READ*,n
```

Sie werden dann am Bildschirm gefragt und tippen die gewünschte Zahl
für N ein; anschließend die Enter-Taste.

Die allgemeine Form der READ-Anweisung ist:

READ*, Eingabe-Liste	
Eingabe-Liste:	eine Folge von Variablennamen, die durch Komma getrennt sind

Beispiele:

```
   READ*, i, y, Var, KK     wartet auf Eingabe von 4 Werten (und Enter)
   READ*                    wartet nur auf Enter
fehlerhaft wäre:
   READ* x        Komma fehlt
   READ*, 10/a    Ausdrücke sind nicht möglich (jedoch bei PRINT*, )
```

Regeln für die Schreibweise der Werte für READ:

1. Daten sind durch Komma oder Leerzeichen voneinander getrennt
2. Daten werden nach den Regeln für Konstante geschrieben.
3. Reicht der Platz einer Zeile nicht für alle Daten, wird automatisch eine neue Zeile eingelesen, bis alle Variablen der Eingabe-Liste mit Werten versorgt sind.
4. Stehen in 1 Zeile mehr Werte, als in der Eingabe-Liste gebraucht werden, werden diese ignoriert. Sie stehen auch für ein weiteres READ nicht mehr zur Verfügung
5. Ein / beendet die Dateneingabe (insbes. am Schirm)

Beispiele für die Dateneingabe (am Ende jeder Zeile Enter drücken!):

```
Anweisung:                      READ*,x,y,i
Daten:                          2.0E4, -7.1 67

Anweisung:                      READ*,x,y,i
Daten in mehr als 1 Zeile:           2.0E4
                                  -7.1          67

2 mal READ hintereinander:      READ*,x,y,i
                                READ*,j
Daten:                          2.0E4
                                     -7.1
                                67.84     25
                               -18   738
```

J erhält den Wert -18; 25 und 738 werden ignoriert; warum?
Erklärung: Das 1. READ liest 3 Werte, also bis 67.84; das 2. READ geht auf eine neue Zeile, ignoriert also 25; ein weiteres READ würde wieder auf eine neue Zeile gehen und 738 ignorieren.

2 Variable, nur 1 Wert (ein / beendet die Dateneingabe):

```
                              READ*,i,x
                              PRINT*,i,x
Daten:                        3    /

Das PRINT liefert             3    ????????    (2. Wert undefiniert)
```

Der Vorteil des READ anstelle einer Zuweisung im Programm ist der, daß das Programm für verschiedene Werte ein Ergebnis liefert (z.B. das Fakultätprogramm für jeden beliebigen eingetippten Wert).

Regel:
Sobald das Programm auf die Anweisung "READ*,..." trifft, hält es an und wartet auf die Eingabe der Daten. Daher sollten Sie vor jedem READ* mit PRINT* angeben, welche Daten erwartet werden.

Umlenken der Standard-Ein/Ausgabe

Wenn man mehr als 24 Ergebniszeilen erzeugt hat, so verschwinden die ersten Zeilen über den oberen Rand weg. WATFOR bietet die Option, die Ergebnisse statt auf den Bildschirm in das Listing-File zu schreiben. Dieses kann man dann editieren und die Ergebnisse ansehen oder auch ausdrucken. Starten Sie das Programm mit
RUN/NOXT oder RUN/NOEX/NOXT bzw. Alt_F3 (programmierte Taste)

So ist jedoch kein Bildschirmdialog möglich, weil z.B. die Aufforderung PRINT*,'bitte ... eingeben' ebenfalls im Listing landet (der gewünschte Text erscheint nicht am Schirm, nur im Listing).

Wollen Sie beides, starten Sie mit
RUN/LOgio oder RUN/NOEX/LOgio bzw. Alt_F5 (programmierte Taste)
Jetzt läuft der Dialog am Schirm normal ab; zusätzlich wird aber das Ergebnis eines jeden READ* und jede Bildschirmeingabe im Listing-File festgehalten!

Für die **Eingabe größerer Datenmengen** empfiehlt sich jedoch ein anderes Verfahren: Mit der **OPEN-Anweisung** kann das Programm individuell seine eigene Einstellung der Geräte-Nr. (s.u. Kap. 7) vornehmen; z.B. richtet

```
OPEN(1,FILE='P1.DAT')
```

alle READ(1,*) -Anweisungen auf das File P1.DAT.
Die Anweisung READ(1,*) wird also anstelle von READ*, geschrieben.
Auf diese Weise können die Daten im voraus erstellt werden (Edit auf das File), sodaß man sie nicht immer wieder am Schirm eintippen muß.
Als Geräte-Nr. sind alle Zahlen von 0 bis 99 möglich.
Verwenden Sie jedoch dabei 5 und 6 nicht, da diese Geräte-Nr. für die Bildschirm- Ein- und Ausgabe reserviert sind (Näheres s.u. Kap. 7).

Umgekehrt schreibt WRITE(2,*) in ein File (s.o. OPEN); das bedeutet aber, daß alle etwa schon vorhandenen Daten des Files gelöscht werden!

Ist das File des angegebenen Namens auf Disk nicht vorhanden, so wird es automatisch angelegt.

Soll der Inhalt eines Files beim WRITE erhalten bleiben, so werden die Daten am Ende angefügt (Vorgriff auf den Fortran90-Standard) nach einem

```
OPEN(2,FILE='P1.ERG',ACCESS='APPEND')
```

Eingebaute Funktionen

Die "INTRINSIC" Funktionen liefern für alle zulässigen Typen von Argumenten automatisch das Ergebnis im richtigen Typ ("Polymorphismus"). Im folgenden bedeutet I / R / D / C / Z / L / c ein Argument vom Typ INTEGER / REAL / REAL*8 (REAL doppelt genau) / COMPLEX / COMPLEX*16 (COMPLEX doppelt genau) / LOGICAL / CHARACTER.

	Typ Argument (||||||)	Typ Resultat (x: wie Typ Argument)	
Typ-Umwandlungen:			
1. REAL(x)	IRDCZ	R	Umwandlung nach REAL
2. DBLE(x)	IRDCZ	D	Umwandlung nach REAL*8(DOUBLE PRECISION)
3. CMPLX(x)	IRDCZ	C	Umwandlung nach COMPLEX
4. DCMPLX(x)	IRDCZ	Z	Umwandlung nach COMPLEX*16 (DOUBLE PRECISION COMPLEX)
5. INT(x)	IRDCZ	I	Umwandlung nach INTEGER
6. NINT(x)	RD	I	Umwandlung nach INTEGER mit Rundung
7. AINT(x)	RD	x	ganzz. Anteil betragsmäßig <= (Nachkommastellen abgeschnitten)
8. ANINT(x)	RD	x	dass. mit Rundung
9. IMAG(x)	CZ	R,D	Imaginärteil
10. CONJG(x)	CZ	x	konjugiert komplexe Zahl
11. CHAR(x)	I	Ch	Umwandlung INTEGER nach CHARACTER
12. ICHAR(x)	c	I	Umwandlung CHARACTER nach INTEGER
Stringverarbeitung (Näheres s. Kap. 8):			
13. LEN(x)	c	I	Länge eines CHARACTER-Strings
14. INDEX(x,y)	c	I	Position von String y in String x
15. LGT(x,y)	c	L	Lexikographisch größer (x > y)
16. LLT(x,y)	c	L	Lexikographisch größer oder gleich
17. LGE(x,y)	c	L	Lexikographisch kleiner
18. LLE(x,y)	c	L	Lexikographisch kleiner oder gleich
Numerische Funktionen:			
19. ABS(x)	IRDCZ	x	Absolutwert
20. MAX(x,...)	IRD	x	Maximum von mehreren Werten dess. Typs
21. MIN(x,...)	IRD	x	Minimum von mehreren Werten dess. Typs
22. MOD(x,y)	IRD	x	1. Argument modulo 2. Argument (Divisionsrest von x / y)
23. SIGN(x,y)	IRD	x	x bekommt das Vorzeichen von y
24. DIM(x,y)	IRD	x	positive Differenz x-y falls x>y,0 sonst
25. DPROD(x,y)	R	D	x * y doppelt genau
26. SQRT(x)	RDCZ	x	Quadratwurzel
Logarithmus, Exponentialfunktion, Trigonometrische Funktionen:			
27. EXP(x)	RDCZ	x	Exponentialfunktion
28. LOG(x)	RDCZ	x	Logarithmus naturalis
29. LOG10(x)	RD	x	Logarithmus zur Basis 10
30. SIN(x)	RDCZ	x	Sinus (Winkel im Bogenmaß)
31. COS(x)	RDCZ	x	Cosinus "
32. TAN(x)	RD	x	Tangens "
33. ASIN(x)	RD	x	Arcussinus "
34. ACOS(x)	RD	x	Arcuscosinus "
35. ATAN(x)	RD	x	Arcustangens "
36. ATAN2(x,y)	RD	x	Arcustangens (x/y) "
37. SINH(x)	RD	x	Sinus hyperbolicus
38. COSH(x)	RD	x	Cosinus hyperbolicus
39. TANH(x)	RD	x	Tangens hyperbolicus

3 Fehlerquellen bei der Arithmetik mit REAL-Zahlen

1. Wir schreiben alle Zahlen im Dezimalsystem; alle Computer (außer Taschenrechner) rechnen aber intern im Dualsystem. Wenn nun die gegebene Dezimalzahl nicht exakt im Dualsystem darstellbar ist, ergibt sich bereits bei der Ein- oder Ausgabe ein Umwandlungsfehler: z.B. ist 1/5 nicht exakt dual darstellbar; allg. alle Dezimalzahlen, die außer den Anteilen 1/2, 1/4, 1/8 ... auch 1/5, 1/10, 1/25 usw. enthalten; z.B.ist 1/5 = 0.00110011... eine unendliche periodische Dualzahl; der Computer kann natürlich nur eine gewisse feste Anzahl von Dualstellen darstellen: die Genauigkeit ist beim Typ REAL 24 bit = 7 Dezimalen, REAL*8: 53 bit = 15 Dezimalen (IEEE-Standard).
2. Durch wiederholte Anwendung der Grundrechenarten werden solche Fehler vergrößert; probieren Sie es aus: Addieren Sie 10 mal 0.1; es müßte am Ende 1.0 herauskommen. **Beispiel**programm:
   (wegen der IF-Anweisung in der Bedeutung von "wenn" s.u. Kap. 3)

```
      r = 0.
      i = 1
      WHILE (i .LE. 10)
*           i mal 0.1 in r aufaddiert, müßte dasselbe sein wie i*0.1
        r = r + 0.1
*Test auf gleich / ungleich:
        IF     (i*0.1  .EQ.  r ) PRINT*, i*0.1, 'gleich',r
        IF     (i*0.1  .NE.  r ) PRINT*, i*0.1, 'ungleich',r
        i = i + 1
      ENDWHILE
```

```
PC: 0.1000000 gleich  0.1000000    WS: 0.1000000015 gleich 0.1000000015
    0.2000000 gleich  0.2000000        0.2000000030 gleich 0.2000000030
    0.3000000 ungleich  0.3000000      0.3000000119 ungleich 0.3000000119
    0.4000000 gleich  0.4000000        0.4000000060 gleich 0.4000000060
    0.5000000 ungleich  0.5000000      0.5000000000 ungleich 0.5000000000
    0.6000000 ungleich  0.6000000      0.6000000238 ungleich 0.6000000238
    0.7000000 ungleich  0.7000000      0.6999999881 ungleich 0.7000000477
    0.8000000 ungleich  0.8000001      0.8000000119 ungleich 0.8000000715
    0.9000000 ungleich  0.9000001      0.9000000358 ungleich 0.9000000954
    1.0000000 ungleich  1.0000000      1.000000000 ungleich 1.000000119
```

Für den einen Computer sind also die 7 Wertepaare paarweise ungleich, was für 0.8 und 0.9 auch sichtbar ist; in allen anderen Fällen führt die Rundung bei der Ausgabe zu einem nur scheinbar exakten Ergebnis!! Zudem sehen wir, daß weder I*0.1 noch R die Zahl 1 exakt erreichen. Auf der Workstation sind die Ergebnisse eher noch verwirrender; klar ist nur, daß beide Computer nur 0.1, 0.2 und 0.4 als "gleich" einschätzen.

Alles in allem ein niederschmetterndes Ergebnis?!

Besser hätten wir also geschrieben:

```
*Test: falls Absolutbetrag der Differenz kleiner als .00001 drucke ...
       IF( ABS(i*0.1 - r) .LT. .00001) PRINT*,i*0.1', wird mit einem
    & Fehler von weniger als 0.00001 erreicht'
```

Regel: REAL-Zahlen nie auf Gleichheit oder Ungleichheit prüfen
also nicht IF (A.EQ.B) oder IF (A.NE.B)
(es sei denn, man weiß genau, daß beide ganze Zahlen sind)
statt dessen IF(ABS(A-B) .LT. 0.0001)
--> 0.0001 stellt eine relativ zum Testwert B kleine Zahl dar,
etwa 3-4 Zehnerpotenzen kleiner als A, falls A einfach genau
etwa 10 Zehnerpotenzen kleiner als A, falls A doppelt genau

Man kann davon ausgehen, daß die Anwendung von 1000 Rechenoperationen auf dieselbe Variable einen Verlust von 1 bis 2 Dezimalen mit sich bringt, im ungünstigsten Fall sogar 3 Dezimalen, wenn die Fehler nicht statistisch verteilt sind, sodaß sich diese z.T. wieder ausgleichen, sondern alle Fehler in dieselbe Richtung gehen wie im Beispiel oben.

3. Fehler 3. Art:
Betrachten Sie die beiden Zeilen
a) PRINT*, 2.0D16 - 2.0D16 + 1.0
b) PRINT*, 2.0D16 + 1.0 - 2.0D16
Mathematisch ist das Ergebnis dasselbe; der Computer rechnet jedoch nur mit 15 Dezimalen = doppelte Genauigkeit; jeder | soll eine der vorhandenen Dezimalen markieren:

```
    ||||||||||||||||                ||||||||||||||||
a)  20000000000000000.0       b)    20000000000000000.0
 -  20000000000000000.0          +                  1.0
    --------------------          --------------------
                     0.0            20000000000000000.0
 +                   1.0          - 20000000000000000.0
    --------------------          --------------------
                     1.0                            0.0
```

Die Addition von 1.0 im Fall b) hat keine Wirkung, da nur 15 Dezimalen verwendet werden. Es kann also jedes Ergebnis, das etwa bei 1 liegt, durch Addition von 1.0E16 oder mehr und nachfolgender Subtraktion desselben Betrages zerstört werden!!
Diesen Effekt nennt man **Auslöschung**; er ist bei den Numerikern ungeheuer gefürchtet, weil mit **einem** Schlag beliebig viele Dezimalen an Genauigkeit verschwinden können, ohne daß der Computer davon Meldung macht (eine in meinen Augen unverzeihliche Nachlässigkeit der Computerhersteller gegenüber nichtkaufmännischen Anwendungen).
Daher die (meist kaum exakt zu befolgende) Regel:

> Man vermeide, wenn irgend möglich, daß Zwischenergebnisse um mehrere Zehnerpotenzen größer als das Ergebnis sind, falls dabei Subtraktionen dieser größeren Beträge auftreten können. Sonst Gefahr der Auslöschung beliebig vieler Dezimalen.

Multiplikation und Division sind im Vergleich relativ harmlos.

Aufgaben

1. Berechnen Sie den Wert von 12-(6-i) für i=1,2,...,12. Es sollten 12 Zeilen Ausgabe erzeugt werden, nämlich i und der entspr. Wert.
2. Tabellieren (berechnen und ausdrucken) Sie x und y aus der Formel

$$y = \frac{x^3 - 4x^2 + x - 3}{|x + 2|}$$

für b) x=-1,-3,...,-11,-13
c) x=10,9,...,-1

3. Lesen Sie X und N ein. Drucken Sie X, X/2, X/4,...,x/2**N
4. Berechnen Sie die Summe der ersten N ganzen Zahlen der ersten N Quadratzahlen, ..., Summe der N**5 für N=20 mit Hilfe der Formeln
   a) Summe der N: n(n+1) /2
   b) Summe der N**2: n(n+1)(2n+1) /6
   c) Summe der N**3: n**2 (n+1)**2 /4
   d) Summe der N**4: n(n+1)(2n+1)(3n**2+3n-1) /30
   e) Summe der N**5: n**2 (n+1)**2 (2n**2+2n-1) /12
5. Eine Näherungsformel für die Quadratwurzel für x im Bereich von .1 ... 10 ist (1+4x)/(4+x). Berechnen Sie die Näherung für x= .5,1.,...,9.5, vergleichen Sie mit dem wahren Wert und berechnen Sie den prozentualen Fehler.
6. Für den dekadischen Logarithmus im Bereich .1 ... 1.:
   log10 x = -.076 + .281x - .238 / (x + .15)
   Für x=.2,.4,...,1. berechnen Sie den Absolutwert der Differenz der Näherung mit dem richtigen Wert.
7. Gesucht ist die kleinste Zahl der benötigten Geldscheine, um die Summe von 1575.00 DM auszuzahlen (eine Schleife ist nicht nötig).

Kapitel 3: Entscheidungen und Steueranweisungen

Fragen

1. Wie wird eine Bedingung als wahr oder falsch erkannt, wie wird sie zur Steuerung (engl. control) des Programmablaufs verwendet?
2. Wenn es mehrere verschiedenen Fälle gibt, wie stellt man es an, die entsprechenden verschiedenen Programmstücke auszuführen?
3. Wie können Anweisungen in einer anderen als der normalen, der sequentiellen Reihenfolge ausgeführt werden?
4. Welche weiteren Möglichkeiten gibt es, eine Entscheidung zu treffen?

Logische Ausdrücke

Logische Entscheidungen werden häufig getroffen, indem etwa X .LE. 3 geprüft wird mit dem Ergebnis wahr oder falsch. Logische Ausdrücke können folgende Vergleichs-Operatoren enthalten:

.LT.	.LE.	.EQ.	.GE.	.GT.	.NE.	für
kleiner	kleiner oder gleich	gleich	größer oder gleich	größer	ungleich	

Logische Operatoren haben geringere Priorität als die arithmetischen Operatoren (+,-,* usw.); d.h. als erstes wird gerechnet, dann erst verglichen; z.B. ist `IF ( x+1 .LE. 2+3 )`
gleichbedeutend mit `IF ( (x+1) .LE. (2+3) )`

Der IF-Block

Beispiel 3.1

Berechnung der Werte von y=1/(x-2) für x=0,1,...,10
Für x=2 wäre eine Division durch 0 zu vermeiden.
Also: falls x=2 dann drucke "y ist undefiniert für x=2"

```
      IF( x.EQ.2 ) THEN
        PRINT*, 'y ist undefiniert für x=2'
      ENDIF
```

Algorithmus:
NB: die **GOTO-Anweisung** "GOTO n" setzt bei der Anweisung mit Nr. n fort, n steht in den Spalten 1-5 und kann 1 bis 5 Ziffern enthalten.

```
  1. setze x=0                        x = 0.
  2. wenn x >= 10 fertig           20 IF ( x .GT. 10.0) GOTO 30
  2.1. falls x=2                      IF ( x .EQ. 2.0) THEN
                drucke ...              PRINT*,'y ist undefiniert: x=',x
  2.2. sonst                          ELSE
            berechne y=...              y = 1.0/(x-2.0)
                                      ENDIF
  2.3. addiere 1 zu x                   x = x + 1.0
  2.4. wiederhole ab 2.               GOTO 20
  3. ende                          30 END
```

Zur IF-Anweisung IF (...) THEN gehört mindestens noch die Anweisung ENDIF(evtl. auch ELSE) dazu. Deshalb nennt man das Ganze den IF-Block.

IF Kurzform: `IF ( Bedingung ) Anweisung`
anstelle von

```
IF ( Bedingung ) THEN
   Anweisung
ENDIF
```

z.B. (s.o.) `IF ( X .GT. 10.0 ) GOTO 30`

IF - THEN - ELSE allgemeine Form:

```
IF - THEN - ELSE (IF-Block):   Programmfluß falls log.Ausdruck
                                        true       false
IF ( logischer Ausdruck ) THEN           T           F
   ...  Anweisungen fuer den       ...|<<<<          :
   ...  Fall true                  ...|              :
   ...                             ...|>>>>          :
ELSE                                  :             :
   ...  Anweisungen fuer den          :     ...|<<<<
   ...  Fall false                    :     ...|
   ...                                :     ...|>>>>
ENDIF                                 :             :
   ...                             ...<<<<<    ...<<<<<
```

Die Anweisungen IF, ELSE und ENDIF stehen senkrecht übereinander;
die inneren Anweisungen eingerückt.

Beispiel 3.2

Welcher Wert wird durch das folgende Programm ausgegeben?

```
     i = 2
     IF(i*2 .LT. -2*i) THEN
       PRINT*,i-3
     ELSE
       j = i + 5
       PRINT*,j
     ENDIF
```

Der logische Ausdruck ist falsch, die THEN-Gruppe wird ignoriert: 7

Beispiel 3.3

Es sind die folgenden 4 Intervalle definiert:

```
<.........|............|............|.........>
          0           15           30
   I1: x<=0  I2: 030
```

Gegeben ein bel. Wert x; gesucht das Intervall, in welchem x liegt.

Methode 1: Intervallhalbierung

Wir wählen einen Wert, der unterscheidet zwischen einerseits
I1 und I2, andererseits I3 und I4:

```
1.     Lies x
2.     Wenn x<=15 dann
 .1        ist x in I1 oder in I2
       sonst
 .2        ist x in I3 oder in I4
3.     Drucke x und das Intervall
4.     Ende
```

Die Unterscheidung zwischen I1 / I2 und I3 / I4 geschieht wieder
mit je einem IF THEN ELSE; also:

```
1.     Lies x                              READ*,x
2.     Wenn x<=15 dann (I1 oder I2)        IF(x.LE.15) THEN
2.1        wenn x<=0 dann                    IF(x.LE.0.0) THEN
   .1         ist x in I1                      i=1
              sonst                          ELSE
   .2         ist x in I2                      i=2
                                             ENDIF
2.2    sonst (I3 oder I4)                  ELSE
           wenn x<=30 dann                   IF(x.LE.30.0) THEN
   .1         ist x in I3                      i=3
              sonst                          ELSE
```

```
     .2         ist x in I4                        i=4
                                                 ENDIF
  3.     drucke x und das Intervall              PRINT*,x,' liegt in I',i
  4.     Ende                                    END
```

Methode 2: Salamitaktik

Wir sehen zuerst, ob x in I1 liegt; dann ob in I2 usw.:

```
  1.     Lies x                                  READ*,x
  2.     wenn x<=0 dann                          IF(x.LE.0.0)THEN
   .1        ist x in I1                           i=1
   .2    sonst (I2 oder I3 oder I4)              ELSE
             wenn x<=15 dann                       IF(x.LE.15.0)THEN
     .1         ist x in I2                          i=2
     .2      sonst (I3 oder I4)                    ELSE
                wenn x<=30 dann                      IF(x.LE.30.0)THEN
       .1          ist x in I3                         i=3
       .2       sonst                                 ELSE
                   ist x in I4                          i=4
                                                 ENDIF

  3.     Drucke x und das Intervall              PRINT*,x,' liegt in I',i
  4.     Ende                                    END
```

Das Einsetzen von weiteren IF-Blöcken in einen IF-Block nennt man Schachtelung. Die Tiefe der Schachtelung ist in unserem Beispiel maximal 3.
Fortran bietet die Möglichkeit, im obigen Beispiel die Schachtelung zu vermeiden:

Methode 3: ELSEIF - sonst wenn:

```
    1.     Lies x                                  READ*,x
    2.     wenn x<=0 dann                          IF(x.LE.0.0)THEN
     .1        ist x in I1                           i=1
     .2    sonst wenn x<=15 dann                   ELSEIF(x.LE.15.0)THEN
                 ist x in I2                         i=2
     .3    sonst wenn x<=30 dann                   ELSEIF(x.LE.30.0)THEN
                   ist x in I3                       i=3
     .4    sonst                                   ELSE
                   ist x in I4                       i=4
                                                   ENDIF
    3.     Drucke x und das Intervall              PRINT*,x,' liegt in',i
    4.     Ende                                    END
```

Mit ELSEIF können beliebig viele alternative Fälle geprüft werden; im ganzen IF-Block wird dann genau 1 Aktion ausgeführt (wenn keine der IF- oder ELSEIF-Bedingungen wahr ist, die ELSE-Aktion). Die verschiedenen Aktionen sind logisch durch ein **exklusives oder** verbunden.

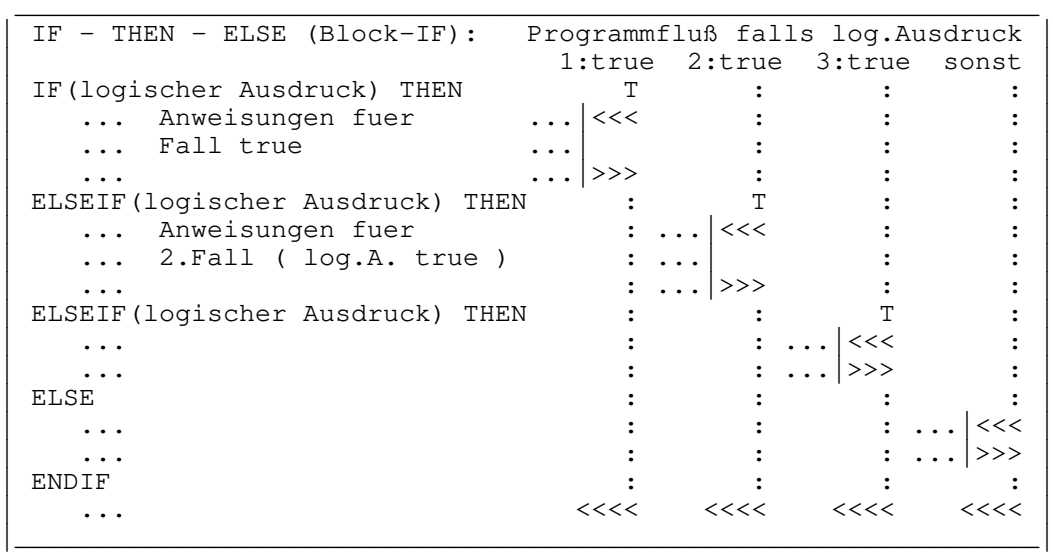
Wenn das ELSE ... weggelassen wird, dann gilt nicht mehr, daß genau e i n e Aktion ausgeführt wird. Dies ist aus einem wichtigen Grund nicht empfehlenswert:
Eine Vielzahl von Fehlern in Computerprogrammen kommt daher, daß der Programmierer logische Bedingungen übersehen hat oder irrtümlich glaubt, diese könnten nie auftreten. Mit ELSE kann man dies ausschließen:

```
      IF( Bedingung )THEN
          ...
      ELSEIF( Bedingung )THEN
          ...
      ELSE
          PRINT*,'darf nicht sein. wenn doch - Programmierfehler'
      ENDIF
```

Schreiben Sie also in jedem Fall ein ELSE. Wenn Sie sicher sind, daß im ELSE-Fall keine Aktion eintreten soll, so dokumentieren Sie dies, indem Sie

```
      IF ...
         ...
      ELSE
      ENDIF
```

schreiben. Die ELSE-Aktion ist hier leer.



Übungen

1. Schreiben Sie einen oder mehrere IF THEN ELSE -Blöcke, die das Ergebnis eines Vergleichs des Wertes von N in Worte fassen, z.B.
   für n=2: 'gleich 2'
   n<>2: 'ungleich 2'
   Lösung:

```
   IF(n.EQ.2) THEN
      PRINT*,n,'gleich 2'
   ELSE
      PRINT*,n,'ungleich 2'
   ENDIF
```

2. a) n > MOD(7,n) 'größer'
   <= 'kleiner gleich'
   b) 5 < n <= 15 'zwischen'
   sonst 'außerhalb'
   c) n > 50 oder n < -20 'verstreut'
   N >= -20 und N < 0 'Bereich 2'
   sonst 'Bereich 3'
   d) einmal ohne, einmal mit ELSEs:

```
   N=2                zwei
   N=4                vier
   N=6                sechs
   N=8                acht
   N=10               zehn
```

3. Betrachten Sie die 3 Methoden des obigen Beispiels 3 (4 Intervalle). Nehmen Sie an, daß es gleich wahrscheinlich ist, daß x in einem der 4 Intervalle liegt. Welche der 3 Methoden ist am effizientesten? Betrachten Sie dazu die durchschnittliche Zahl der notwendigen logischen Vergleiche.

Logische Operatoren

Mit .NOT. für "nicht", .OR. für "oder", .AND. für "und" werden logische Ausdrücke gebildet, Klammerung ist ebenfalls möglich; z.B.

```
(R1 .AND. R2 .AND. R3 .AND. R4)   true, wenn alle 4 true sind
(R1 .OR. R2)                      true, wenn auch nur 1 true ist
```

Es soll x ausgegeben werden, falls x innerhalb des Intervalls (-1,+1):

```
   IF( x .GE. -1 .AND. x .LE. 1 ) THEN
      PRINT*,'x=',x
   ELSE
   ENDIF
```

Es soll x, falls außerhalb des Intervalls, ausgegeben werden:

```
   IF( x .LT. -1) THEN
      PRINT*,'x=',x
   ELSEIF( x .GT. 1) THEN
      PRINT*,'x=',x
   ELSE
   ENDIF
```

Man kann aber die beiden Bedingungen auch so zusammenfassen:

```
   IF( x.LT.-1 .OR. x.GT.1 ) THEN
      PRINT*,'x=',x
   ELSE
   ENDIF
```

oder, wenn Sie an die 1. Version denken (drucken wenn x im Intervall) :

```
   IF( .NOT. (x.GE.-1 .AND. x.LE.1) ) THEN
      PRINT*,'x=',x
   ELSE
   ENDIF
```

Prioritäten bei der Auswertung mit logischen Operatoren:

1. Funktionen (von rechts nach links)
2. Klammern (innerste zuerst)
3. Arithmetische Operatoren:
   - 3.1 ** von rechts nach links
   - 3.2 * / von links nach rechts
   - 3.3 + - von links nach rechts
4. Vergleichs-Operatoren (von links nach rechts)
   .LT. .LE. .EQ. .NE. .GT. .GE.
5. logische Operatoren (von links nach rechts)
   - 5.1 .NOT.
   - 5.2 .AND.
   - 5.3 .OR.

Aufgaben

1. Schreiben Sie ein Programm, das einen INTEGER-Wert einliest und die Zahl der Ziffern ungleich 0 bestimmt. Hinweis: dividieren Sie durch 10, 100, 1000 usw. und sehen Sie zu, ob der Rest 0 ist.

2. Lesen Sie 2 Integers N und M. Lesen Sie einen Wert K. Prüfen Sie, ob die letzten K Stellen in M und N dieselben sind. Drucken Sie 'die letzten ... Stellen von ... und ... sind gleich' bzw. '... ungleich'.

3. Lesen Sie die 3 Seitenlängen eines Dreiecks ein. Drucken Sie '0', wenn die Längen kein Dreieck bilden, '1' wenn alle Seiten ungleich sind, '2' bei gleichschenkligem, '3' bei gleichseitigem Dreieck.

4. Man berechne die Wurzel aus 12; sie liegt zwischen 3 (low) und 4 (high); man drucke l und h aus und halbiere das Intervall (3...4), d.h. man nimmt (h+l)/2; nun sieht man, ob das Quadrat dieses Wertes größer oder kleiner als 12 ist; im ersteren Fall nimmt man den Wert als neues h, im anderen als neues l usw., bis der wahre Wert bis auf 0.01 angenähert ist (bis sich h und l um weniger als 0.02 unterscheiden): die Wurzel aus 12 liegt zwischen h und l, wenn wenn h**2 > 12, l**2 < 12 ist. Man vergleiche mit SQRT(12).

5. Der Wochentag kann für ein bel. Datum aus Jahr (J), Monat (M) und Tag (T) mit folgender Formel gefunden werden:

   Wochentag = K MOD 7 mit

$$K = T + 2M + \frac{3M+3}{5} + J + \frac{J}{4} + \frac{J}{100} + 1$$

   Alle Divisionen sind ganzzahlig (Division von Integer-Zahlen).
   Januar und Februar müssen als Monat 13 und 14 des alten Jahres aufgefaßt werden.
   Wochentag 0 ist Montag usw.
   Geben Sie verschiedene Daten (Ihren Geburtstag) an; stoppen Sie mit einem negativen Jahr.

6. Drucken Sie alle ganzen Zahlen zwischen 100 und 500 aus, die lauter verschiedene Ziffern enthalten. Man drucke je 4 Zahlen pro Zeile.

Kapitel 4: Schleifen

Fragen

1. Was ist eine Schleife?
2. Wie sehen die beiden Arten von Schleifen in Fortran aus?

Schleifen und Schleifensteuerung

In Kap. 1 hatten wir Algorithmen, wo eine oder mehrere Anweisungen mehrfach ausgeführt wurden. Diese Art von Wiederholung nennt man eine Schleife. Die Steuerung der Schleife, d.h. die Logik, mit der die Ausführung der Schleife gesteuert wird, dh. unter welchen Bedingungen die Schleife ausgeführt, unter welchen abgebrochen wird, ist Teil der Anweisung, z.B. in (...) bei der Anweisung WHILE (...), oder wird in Form weiterer Bedingung(en) in der Schleife geschrieben, s.u. Wird sie nie abgebrochen, spricht man von einer Endlosschleife.

WHILE-Schleife

Die WHILE-Schleife besteht aus 4 Teilen:
1. Eine Anweisung zum Setzen des Ausgangswertes für den logischen Ausdruck der WHILE-Anweisung.
2. Die WHILE-Anweisung, in Klammern ein logischen Ausdruck. Ist er wahr, werden die unmittelbar folgenden Anweisungen (die Schleife) ausgeführt. Ist er falsch, geht es hinter ENDWHILE weiter.
3. Das Innere der Schleife muß eine Anweisung enthalten, die irgendwann den logischen Ausdruck falsch macht (Beenden der Schleife).
4. ENDWHILE als letzte Schleifenanweisung.

Beispiel 4.1

```
      N = 1                                Setzen Anfangswert
      WHILE ( N .LE. 10 )                  Bedingung für Ausführung
        PRINT*,N                             beliebige Aktion
        N = N + 1                            notwendig für Beenden
      ENDWHILE                             markiert Schleifenende
```

WHILE - Schleife allgemeine Form:

```
   WHILE:                        Programmfluß falls log.Ausdruck
                                          false      true
WHILE ( logischer Ausdruck )              >>>>>      />>>>
   ...  Anweisungen der Schleife              :      /   ...
   ...                                        :      /   ...
ENDWHILE                                      :      /<<<<
...     Fortsetzung                       ...<<<<
```

NB: Die WHILE-Schleife ist ein Vorgriff auf den Fortran90-Standard
DO WHILE (...) ... ENDDO

Übungen

1. Sehen Sie sich die folgenden Programme an. Keines läuft ohne Fehler. Erklären Sie, warum.

a)
```
      x = 1
      WHILE(x .LE. 10)
        PRINT*,x
        PRINT*,3*x+2
      ENDWHILE
      END
```

b)
```
      WHILE(x*2 .GT. 5)
        PRINT*,x
        x = x - 1
      ENDWHILE
      END
```

c)
```
x = 10
WHILE(x**2 .GE. 100)
  x = x + 1
ENDWHILE
PRINT*,x,x**2
END
```

d)
```
x = 1
WHILE(x .LE. 10)
  PRINT*,x
  x = x + 1
END
```

2. Die Regeln für WHILE sind:
   (1) Die Bedingung im WHILE muß vorher gesetzt werden
   (2) Die Bedingung im WHILE muß irgendwann einmal false werden
   (3) Das Innere der Schleife wird abgeschlossen mit ENDWHILE
   Welche der 3 Regeln ist in den 4 Programmen oben verletzt?

DO-Schleife

Oft möchte man eine Schleife entweder eine feste Anzahl oder eine maximale Anzahl von Malen durchlaufen. Dies kommt so häufig vor, daß Fortran eine besondere Anweisung dafür bereitstellt (obwohl alles auch mit der WHILE-Schleife gemacht werden könnte).

Beispiel 4.2

```
      PROGRAM P42
* Beispiel 4.2: Wie Beispiel 1.3 (Kap.1), aber jetzt mit DO-Schleife
      REAL Kapital, Zinsertrag
      INTEGER Jahr
      Kapital = 100
      DO Jahr = 1991, 2000, 1
        Zinsertrag = Kapital * 0.12
        PRINT*,'Kapital',Kapital,'            Zinsertrag',Zinsertrag
        Kapital = Kapital + Zinsertrag
      ENDDO
      END
```

Erklärung:
   Die DO-Anweisung setzt zunächst Jahr auf 1991; dann wird getestet, ob dieser Wert kleiner oder gleich 2000 ist: ",2000" . Wenn also 2000 noch nicht überschritten ist, werden die Anweisungen der Schleife ausgeführt. Dann wird der Wert von Jahr um 1 erhöht: ",1" und der Test wiederholt, bis der Wert von Jahr größer als 2000 ist.

Beispiel 4.3 (Quadratwurzel der positiven Zahl X):

E sei eine Näherung (positive Zahl), z.B. X/2;
neues E nach Newton = Ealt - (Ealt**2 - X) / (2*Ealt)

Algorithmus (für DO formuliert):

```
  1. Lies Wert von X
  2. Setze E = X/2
  3.0 Wiederhole  10 mal
  3.1 Formel für E
  3.2 oben weiter
  4. Ausdruck der Näherung
  5. Ende
```

mit DO:

```
READ*,x
e = x/2.0
DO i=1,10,1
  e = ...
ENDDO
PRINT*,e
END
```

mit WHILE:

```
READ*,x
e = x/2.0
i = 1
WHILE(i.LE.10)
  e = ...
  i = i + 1
ENDWHILE
PRINT*,e
END
```

Vergleichen Sie die logisch gleichwertigen Programme mit DO und mit WHILE; Sie sehen die kompaktere Formulierung der DO-Anweisung, wenn es sich um einfaches Herauf- (oder Herunter-) -Zählen handelt.
Die WHILE-Schleife kann demgegenüber allgemeine logische Bedingungen zur Schleifensteuerung heranziehen.

Die DO-Anweisung DO i=1,10,1 besagt:
Wiederhole alle Anweisungen bis ENDDO wobei der Zähler i anfangs 1 ist und mit der Schrittweite 1 bis 10 läuft.

Welche Werte werden mit

```
DO n=4,24,5
  PRINT*,n
ENDDO
```

ausgegeben?

Es sind die Werte 4,9,14,19,24: Anfangswert von n ist 4, Schrittweite 5, Ende, wenn 24 überschritten, d.h. bis einschließlich 24.

Allgemeine Form der DO-Schleife:

```
DO-Schleife ( DO-Block ):
      DO Variable = Anfangswert, Endwert, Schrittweite
        ...
        ... (Inneres der DO-Schleife)
        ...
      ENDDO

   Regeln:
1. Beim 1. Durchgang hat die DO-Variable den obigen Anfangswert
2. Falls der Anfangswert größer als der Endwert (wenn die
   Schrittweite positiv) bzw. kleiner (Schrittweite negativ),
   wird die DO-Schleife nicht ausgeführt, weiter hinter ENDDO
3. Am Ende der DO-Schleife (bei ENDDO) wird zur DO-Variablen
   die Schrittweite addiert.
4. Weiter wie unter 2. (Schleife wiederholen oder abbrechen)
5. Die Schrittweite (mit vorangehendem Komma) kann entfallen
   und wird in diesem Fall mit 1 angenommen.
6. Die DO-Variable sowie Anfangswert, Endwert und Schrittweite
   (die DO-Steuergrößen) dürfen innerhalb der DO-Schleife
   nicht verändert werden.
Besonderheit:
   Die DO-Parameter sollen vom Typ INTEGER sein
```

NB: Die Schreibweise DO .. ENDDO ist ein Vorgriff auf den Fortran90-Standard.

Beispiel 4.4: Zähler läuft rückwärts:

```
DO n=10,0,-1    setze Zähler = Anfangswert; schon fertig?
  PRINT*,'n=',n         Aktion
ENDDO            erniedrige Zähler um Schrittweite; fertig? s.o.
```

Beispiel 4.5: STOP-Anweisung: Programmende innerhalb der DO-Schleife:

```
                 3    2
für x=1., 1.5, 2., 2.5, ..., 10. Falls irgendwann durch 0 dividiert
werden müßte, soll x ausgegeben und das Programm beendet werden.
      REAL x,y,z
*stattDO x=1.0, 10.0, 0.5 schreiben wir
      DO i=  2,   20,   1
* und berechnen x mittels Division durch 2:
        x = REAL(i) / 2.0
        z = (5.0*x**3 - x**2 + 4.0*x -7.0)
        IF ( z .EQ. 0.0) THEN
          PRINT*, 'für x=',x,' Division durch 0 -> Programmende'
          STOP
        ENDIF
        PRINT*, 'x=',x,' y=',1.0/z
      ENDDO
      END
```

Übungen

1. Betrachten Sie folgendes Programm, ehe Sie die Fragen beantworten:

```
      DO j = 5 , 100 , 5
        k = j**2
        PRINT*,k
      ENDDO
```

   Fragen: a) welcher Wert wird als erstes ausgegeben?
   b) wieviele Werte werden ausgegeben?
   c) was tut das Programm, in Worten ausgedrückt?
   d) welche Änderungen sind nötig, wenn man die Werte 25,36,49,64,81,100,121,144 haben will?

2. Man zähle die 5 in jeder DO-Schleife enthaltenen Informationen auf.
3. Beschreiben Sie in Worten, was DO J=21,55,2 bedeutet.
4. Wie oft werden folgende DO-Schleifen durchlaufen:
   a) DO i=2,5
   b) DO j=14,99,14
   c) DO k=9,7,2
   d) DO Loop=1,1,1
   e) DO Max0=100,203,50
5. Wo liegt der Fehler?

```
      DO i=1,12,2
         IF (...) i=i+1
      ENDDO
```

6. Geschachtelte DO-Schleifen: Matrix-Multiplikation (s. Kap. 5):

```
      REAL A(10,10),B(10,10),C(10,10)
*  C=A*B mit Matrizen A,B,C
      DO i=1,10
        DO j=1,10
          C(i,j)=0.0
          DO k=1,10
            C(i,j)=C(i,j) + A(i,k) * B(k,j)
      ENDDO
```

   Schreiben Sie das Programm so um, daß die Summierung in einer einfachen Variablen ausgeführt wird und C(i,j) erst nach Ende der Summierung seinen Wert zugewiesen bekommt (dies ist eine beliebte Methode, das Programm schneller zu machen).

Mehrere Endebedingungen in der Schleife

Zusätzlich zum automatischen Test der Endebedingung der WHILE- bzw. DO-Schleife können weitere Tests auf Abbruch folgen.

DO und WHILE mit mehreren Endebedingungen:

```
       DO Var=AWert,TWert,Schritt          bzw. WHILE(Bed.)
         ...                                    .
         IF (Endebedingung1) GOTO Endnr         wie links
         ...                                    .
         IF (Endebedingung2) GOTO Endnr         wie links
           ...                                  .
       ENDDO                                bzw. ENDWHILE
 Endnr ...
```

   Regeln:
1. Die Schleife kann von einer beliebigen inneren Stelle aus mit einer Endebedingung verlassen werden (auch mehrere Endebedingungen sind möglich).
2. Die zusätzlichen Endebedingungen verweisen auf die Anweisung unmittelbar hinter ENDDO bzw. ENDWHILE, evtl. auch auf eine wenige Zeilen tiefer stehende Anweisung

Beispiel 4.6: Ausnahmebedingung in einer Schleife:

Problem:
für x=1.0,2.0,...,10.0, jedoch nur bis zu der Stelle, wo der Nenner 0 wird: dann soll eine Meldung gegeben und das Programm beendet werden.

Algorithmus:

```
1. Für I=1 bis 10 tue folgendes
1.0 Schleifensteuerung
1.1 setze X=I (X ist Real, I Integer!)
1.2 berechne Nenner nach der Formel
1.3 WENN Nenner=0,
1.3.1              gib Meldung
1.3.2              Programmstop (Ausnahmefall)
1.4 ANSONSTEN
1.4.1         drucke Ergebnis
1.5 nächster Wert I; weiter bei 1.0
4. Ende
```

```
   Version mit WHILE:                          Version mit DO:
     x = 1.0                      (1.0)         DO i=1,10
     WHILE( x .LE. 10.0)          (1.0)           x = REAL(I)
       Yn= ...                    (1.2)           Yn = ...
       IF( Yn .EQ. 0.0 ) THEN     (1.3)           IF(Yn .EQ. 0.0) THEN
         PRINT*,'Nenner ist 0'    (1.3.1)           PRINT*,'Nenner ist 0'
         STOP                     (1.3.2)           STOP
       ELSE                                        ELSE
         PRINT*, x, 1.0/Yn        (1.4.1)           PRINT*, x, 1.0/Yn
       ENDIF                                       ENDIF
       x = x + 1.0                (1.5)
     ENDWHILE                     (4.)          ENDDO
     END                                        END
```

Am Ende von Kap. 5 finden Sie weitere Ausführungen zu diesem Thema.

Geschachtelte Schleifen

Beispiel 4.7:

Problem:

$$\text{berechne } z = \frac{x - 2xy + y}{(x + 4)(y + 20)} \quad \text{für } x=1,1.5,2,...,5;\ y=30,28,26,...,4$$

Algorithmus:

```
  1. Setze Schleife an für x=1,...
  1.1 Setze Schleife an für y=4,...
  1.1.1 Berechne z
  1.1.2 Drucke z
  2. Ende
```

Programm Version 1:

```
 * geschachtelte Schleifen: X geht ab 1.0 bis 5.0 Schritt 0.5
       DO I = 2,10,1                                 -----------------|
         X = I * 0.5                                           äußere |
         DO J = 30,4,-2                              --------|        |
           Y = REAL(J)                                       |        |
           OBEN = X**2 - 2.0*X*Y + Y**2                      |        |
           UNTEN = ( X + 4.0 ) * ( Y + 20.0 )   innere       |        |
           Z = OBEN / UNTEN                         Schleife|        |
           PRINT*,'X=',X,' Y=',Y,' Z=',Z                     |        |
         ENDDO                                       --------Schleife|
       ENDDO                                         -----------------
       END
```

Ändern Sie den Nenner zu (x - 4) * (y - 20): für x=4 und y=20 ist der Nenner 0, die Division unmöglich. Geben Sie eine Meldung aus.

Programm Version 2:

```
*  geschachtelte Schleifen mit neuem Nenner (x-4)*(y-20)
      DO I = 2,10,1
        X = I * 0.5
        IF ( ABS ( X - 4.0 ) .LT. 0.0001 ) THEN
          PRINT*,'bei x=' X,' ist der Nenner praktisch 0'
        ELSE
        DO J = 30,4,-2
          Y = REAL(J)
            IF ( ABS ( Y - 20.0 ) .LT. 0.0001 ) THEN
              PRINT*,'bei y=' Y,' ist der Nenner praktisch 0'
            ELSE
              OBEN = X**2 - 2.0*X*Y + Y**2
              UNTEN = ( X - 4.0 ) * ( Y - 20.0 )
              Z = OBEN / UNTEN
              PRINT*,'X=',X,' Y=',Y,' Z=',Z
            ENDIF
          ENDDO
         ENDIF
      ENDDO
      END
```

Aufgaben

1. Tabellieren Sie die Funktion (x**2 - x + 7)/(x**2 +2).
   Schreiben Sie DO-Schleifen für folgende x-Werte:
   a) 1,2,3,...,20 c) 0,-1,-2,...,-8
   b) 3,6,9,...,24 d) 12,10,8,...,-10

3. für x=-1,-3/4,-1/2,...,1. Übergehen Sie die Fälle mit Nenner = 0.

4. Eine verbesserte Näherung eines Näherungswertes x0 für 1/Wurzel x
   ist X(n) = 0.5 * X(n-1) * (3 - X * X(n-1)**2).
   Berechnen Sie 1/Wurzel 2 mit Anfangswert 0.4 in 5 Schritten.

5. Nehmen Sie an, daß der Computer nicht dividieren kann. Eine Methode,
   den Wert 1/x nach und nach anzunähern, ist, x0, x1, ... zu berechnen
   mit $X_n = X_{n-1} (2 - A*X_{n-1})$, x0 ist ein bel. Anfangswert.
   Schreiben Sie ein Programm für 1/3 (A=3) und verwenden Sie 0.3
   als Anfangswert. Geben Sie für jede Iteration den x-Wert aus.

6. Man berechne die ersten 15 Fibonacci-Zahlen aus 2 Startzahlen, die
   eingelesen werden; die weiteren ergeben sich jeweils als Summe der
   beiden vorangegangenen. Man lese die Zahlen 0,1 ein.
   Diese Programm schließe man in eine 2. äußere Endlosschleife ein,
   die immer wieder nach 2 Startzahlen fragt. Abbruch mit Strg_z.
   Hinweis: Verwenden Sie WHILE(.TRUE.) .. ENDWHILE als Endlosschleife.

7. Erweitern Sie Aufgabe 8 so, daß bei jedem Durchlauf die Quotienten
   je 2 aufeinanderfolgender Zahlen ausgegeben werden (Typ REAL).

8. Auf einer Party mit n Paaren zieht jede Frau den Namen eines Mannes
   aus einem Hut. Die Wahrscheinlichkeit, nicht den Namen des eigenen
   Partners zu ziehen, ist $P_n = ((n-1)/n) P_{n-1} + (1/n) P_{n-2}$
   wo P0 = 0 und P1= 0.5. Berechnen Sie P für n=2,3,...

9. Lesen Sie einen Wert n=10 ein. Drucken Sie alle Tripel von positiven
   ganzen Zahlen mit Summe n; dasselbe für alle Tripel nicht-negativer
   ganzer Zahlen mit Summe n. Die Tripel sollen durchnumeriert werden.

9. Tabellieren Sie die Funktion $z = (x^2 - 2xy + 3y^3) / (xy+3x+2y+4)$

   für alle Paare x,y mit x=1,2,...,6 und y=1,2,...5.

10. Nehmen Sie an, der Computer könne nur mit 2 multiplizieren oder dividieren (was tatsächlich stimmt!).
    Das Produkt zweier Zahlen kann man dann so bestimmen:
    Man dividiere die kleinere Zahl der Reihe nach durch 2, ohne den Rest zu beachten, gleichzeitig wird die größere mit 2 multipliziert.
    Alle Produkte der größeren Zahl werden aufaddiert, falls die zugehörige Division ein ungerades Ergebnis geliefert hat.

    ```
    Beispiel: 37*65 :  37    65
                       18        130  (18 ist gerade)
                        9   260
                        4        520  (4  ...
                        2       1040  (2  ...
                        1  2080
                          +----
    Ergebnis:             2405
    ```

    Schreiben Sie ein Programm, welches beliebige Zahlenpaare einliest und das Produkt nach diesem Verfahren berechnet.

    NB: Kleinere Microcomputer benötigen Multiplikationsprogramme dieser Art (für Dualzahlen statt wie hier Dezimalzahlen). In größeren Computern sind diese als Maschinencode-Befehl fest eingebaut.

11. Auf der Insel Ho leben zwei Rassen - Gute, die immer die Wahrheit sagen, und Böse, die immer lügen. Bei meiner letzten Reise stellte ich dreien von ihnen, Tom, Dick und Zack, je eine Frage (Antwort in Klammern):
    a) Tom, ist Dick ein Guter? (ja)
    b) Dick, gehören Tom und Zack zur selben Rasse? (nein)
    c) Zack, ist nun Dick ein Guter? (ja)
    Schreiben Sie ein Programm zur Bestimmung der Rasse der drei. Dazu generieren Sie alle 8 möglichen Kombinationen der Zugehörigkeit zu den beiden Rassen (3 DO-Schleifen: Tom,Dick,Zack = true oder false). Prüfen Sie jede Kombination, ob sie so antworten würde wie oben. Dies kann mit nur 1 IF-Abfrage bewirkt werden! Bedenken Sie, daß der Computer nicht denken kann: Sie müssen aus allen theoretisch möglichen Antworten die richtigen Schlüsse ziehen.
    Daraus müssen Sie die IF-Anweisung konstruieren.
    Hinweis: Zur Prüfung, ob 2 logische Bedingungen entweder beide wahr oder beide falsch sind, gibt es den logischen Operator .EQV.

12. Mit dem Newton-Algorithmus zur Bestimmung der Wurzel (Bsp. 4.3):
    Führen Sie für die Wurzel aus 4059.26 8 Iterationen aus.
    Drucken Sie den Wert der Näherung nach jedem Schritt.

13. Die folgende Aufgabe ist mehr für Naturwissenschaftler geeignet:
    Führen Sie ein anderes Endekriterium statt Zahl der Iterationen ein: die Bedingung für die nächste Iteration soll jetzt sein, daß die neue Näherung um nicht mehr als 0.0001 von der alten verschieden ist. Testen Sie mit x=3.0.
    Den Abbruch der Iteration mit dieser Bedingung können Sie mit der QUIT-Anweisung schreiben (Vorgriff auf den Fortran90-Standard, dort heißt es EXIT). QUIT verläßt die Schleife (DO- oder WHILE_Schleife) und macht direkt hinter ENDDO (oder auch ENDWHILE) weiter.
    Setzen Sie vorher den Iterationszähler auf eine bestimmte maximale Zahl. Prüfen Sie, ob das Ende der Iteration durch Erschöpfen der Maximalzahl oder durch Erfüllen des neuen Abbruchkriteriums zustande gekommen ist.

Kapitel 5: Felder

Fragen

1. Manche der bisherigen Probleme waren mühsam zu programmieren; warum?
2. Wie können größere Mengen von Variablen definiert werden?
3. Was versteht man unter der Zuweisung von Anfangswerten?

Beispiel 5.1

Man lese N (max.6) Werte ein und drucke sie in verkehrter Reihenfolge aus. Mit den bisherigen Sprachelementen sieht das so aus (aus Platzgründen stehen mehrere Anweisungen mit ; getrennt auf 1 Zeile):

```
READ*,N
IF(N.GE.1)  READ*,X1; IF(N.GE.2)  READ*,X2
IF(N.GE.3)  READ*,X3; IF(N.GE.4)  READ*,X4
IF(N.GE.5)  READ*,X5; IF(N.GE.6)  READ*,X6
IF(N.GE.6) PRINT*,X6; IF(N.GE.5) PRINT*,X5
IF(N.GE.4) PRINT*,X4; IF(N.GE.3) PRINT*,X3
IF(N.GE.2) PRINT*,X2; IF(N.GE.1) PRINT*,X1
END
```

Bemerkungen:

1. Wir müssen alle Werte einlesen, bevor ausgegeben wird.
2. Für jeden Wert muß eine Anweisung geschrieben werden.
3. Bei sehr vielen Werten wird das Programm enorm lang; wir benötigen ein Verfahren, um die Bezeichnung der Variablen zu verändern.

Idee:

```
READ*,N
DO i=1..N
  READ*, Xi
ENDDO
DO i=N..1
  PRINT*, Xi
ENDDO
```

Fortran:

```
REAL X(5)
READ*,n
DO i=1,n
  READ*, X(i)
ENDDO
DO i=n,1,-1
  PRINT*, X(i)
ENDDO
```

Hier ändert sich der "Name" der Variablen X bei jedem Durchlauf der DO-Schleife.
Wir Schreiben in Fortran "REAL X(5)" und haben damit ein Feld X mit 5 Feldelementen, die X(1), X(2), X(3), X(4), X(5) heißen.
Man sagt auch "eindimensionales Feld" oder auch Vektor. Die Zahl in Klammern heißt Index. Die Angabe 5 in REAL(5) legt die Zahl der verfügbaren Feldelemente von X fest; diese Angabe muß eine positive ganze Zahl (eine Konstante, keine Variable) sein.
Ist die benötigte Zahl der Feldelemente nicht bekannt, setze man die Dimensionierung auf die größte zu erwartende Zahl.
z.B. definiert REAL A(20),B,C(1000) genau 1021 REAL-Variable in 2 Feldern und einer einfachen (skalaren) REAL-Variablen.

Beispiel 5.2

Man berechne die ersten 15 Fibonacci-Zahlen wie in Aufg. 8 von Kap. 4 mit 0,1. Man speichere alle Fibonacci-Zahlen in ein Feld ab.

```
INTEGER F(15)
F(1)=1
F(2)=2
DO i=3,15
  F(i) = F(i-1) + F(i-2)
ENDDO
PRINT*,F
END
```

Neu sind 2 Dinge:
1. In der PRINT-Anweisung wird der Feldname F ohne Index verwendet; so kann man das ganze Feld drucken (auch bei READ möglich).
2. Als Indizes sind beliebige INTEGER-Ausdrücke, wie F(I-1), oder auch komplizierte REAL-Ausdrücke wie F(2.*LOG(X)/2.) erlaubt. Jeder solche Ausdruck wird in den Typ INTEGER umgewandelt und dann als Index verwendet - die Regeln sind wie bei der Zuweisung I = R.

Da die Namen der Feldelemente von F F(1), F(2),...,F(15) sind, muß eine der Zahlen 1 bis 15 herauskommen; kommt z.B. 0 heraus, meldet der Compiler *ERR* SS-04 subscript expression out of range

Weitere **Beispiele**: F(F(4))

Hier ist der Index ein Feldelement desselben Feldes. F(4) ist die 4. Fibonacci-Zahl, also 3; F(F(4)) ist also hier dasselbe wie F(3). Jedoch würde F(F(9)), da F(9)=34 ist, den Fehler "Indexüberlauf" erzeugen (bitte ausprobieren und die englische Meldung verstehen!).

F(F(F(6)-2)-3)):
Dies ergibt F(F(8-2)-3) = F(8-3) = F(5) = 2

Angenommen, in F und G sind je 20 Elemente gespeichert; wir schreiben INTEGER F(20), G(20) und wollen vergleichen, ob dieselben Zahlen gespeichert sind. Man kann jedoch nicht schreiben: IF(F .EQ. G) .., sondern muß die Feldelemente einzeln ansprechen, etwa so:

```
        DO i=1,20
          IF ( F(i) .NE. G(i) ) THEN
            PRINT*,'F(',i,') ist ungleich G(',i,')'
          ENDIF
        ENDDO
```

Übungen

1. Betrachten Sie folgendes Programm:

```
        REAL A(4)
        DO i=1,4
          A(i) = REAL(i)*2.+3.
          PRINT*,A(i)
        ENDDO
        END
```

Fragen:
a) Wird jedem Element des Feldes ein Wert zugewiesen? Warum?
b) Wieviele der ausgegebenen Zahlen sind Primzahlen?
c) Warum erscheint keine Fehlermeldung, obwohl die Werte der Feldelemente außerhalb des Bereichs 1 bis 4 liegen?

2. Betrachten Sie die folgenden Anweisungen:

```
        INTEGER BAC(5)
        DO JKL=1,5
          BAC(6-JKL) = JKL
        ENDDO
```

Fragen:
a) Welcher Wert wird BAC(4) zugewiesen?
b) Welche Variable hat den Wert 4?
c) Welchen Wert hat
   aa) BAC(2)/3-BAC(2)
   bb) MOD(BAC(4),3) + MOD(BAC(3),4)
   cc) BAC(4)**BAC(3)**BAC(4)
   dd) BAC(3) - BAC(2) .LT. BAC(1)/4 (wahr oder falsch)
   ee) BAC(BAC(BAC(BAC(1))))

3. Finden Sie den Fehler in folgenden Zeilen; A und HAPPY seien mit
   INTEGER A(2); REAL HAPPY(3) deklariert.
   a) A(2) = HAPPY(4)
   b) X = HAPPY
   c) J = INT(A(1))
   d) A = MOD(3.,2.5)

4. Beschreiben Sie ein Feld aus ganzen Zahlen mit Namen Feld mit einer DO-Schleife und einer zusätzlichen Anweisung mit den Werten
   a) 2,5,8,11,14,17
   b) -5,-4,-3,-2,-1
   c) -5,-7,-9,-11,-13
   d) 1,3,7,15,31,63
   e) 1,0,1,0,1
   f) 1,0,-1,1,0,-1

Ein- und Ausgabe von Feldern

Angenommen ein Feld REAL X(6) soll mit Daten versehen werden. Dazu gibt es 4 Methoden:

```
1.   READ*,X(1),X(2),X(3),X(4),X(5),X(6)    1 READ - mind. 1 neue Zeile
2.   READ*,X                                 Kurzform für 1.
3.   DO i=1,6
       READ*,X(i)            hier wird READ 6 mal ausgeführt,
     ENDDO                   also werden mind. 6 Zeilen eingelesen
4.   READ*,(X(i),i=1,6)            äquivalent zu 1 oder 2.
```

Letzteres heißt **"implizite DO-Schleife"**. Die Regeln sind dieselben wie für DO, nur daß anstelle von DO und ENDDO (und) stehen.
Allgemeine Form:
Liste kann eine oder mehrere Variable, Ausdrücke oder Felder enthalten.
Schrittweite kann entfallen und wird dann als 1 angenommen.

Übungen

1. D und T seien REAL D(16),T(16). Man bestimme Anzahl und Anordnung der ausgegebenen Werte:
   a) PRINT*, T,D
   b) PRINT*, (T(i),D(i),i=1,16)
   c) PRINT*, ((D(i+j-1), j=1,4), i=1,13,4)
   d) PRINT*, ((T(i+j-1), j=1,13,4), i=1,4)
2. Welche Werte bekommen die Elemente von W?

```
INTEGER W(5)
W(3) = 1
DO n = 1,2
  W(n+3) = 3-n
  W(3-n) = 3+n
ENDDO
```

3. Formulieren Sie 2 Anweisungen, mit denen 80 Zeichen in 1 Zeile geschrieben werden, und zwar je 10 Nullen, Einser, Zweier,..., Neuner. Verwenden Sie 2 implizite DO-Schleifen. Kann man dies auch mit einer erreichen? Oder ohne implizite DO-Schleife?
   Schreiben Sie statt PRINT*,... WRITE(*,'(80I1)')...
   dies vermeidet die störenden Leerzeichen (s.u. Kap.7).

2-dimensionale Felder - Tabellen

Wir wollen eine Umfrage in 6 Häusern einer Straße machen und nach der Zahl der Vorschulkinder, Grund/Haupt-, Real/Gymnasialschulkinder und Erwachsener fragen. Die Ergebnisse tragen wir in eine Tabelle ein:

	Vorschule	Grund/Hauptschule	höhere Schule	Erwachsene
Haus 1	0	1	2	2
Haus 2	2	0	0	2
Haus 3	0	0	0	1
Haus 4	0	2	0	2
Haus 5	1	0	1	2
Haus 6	0	2	3	3

Eine rechteckige Tabelle (hier 6 mal 4 Werte) nennt man auch Matrix.
Jedes Element wird durch Angabe der Zeile und Spalte angesprochen;

z.B. steht in Position (5,2) die Zahl der Grund/Hauptschüler in Haus 5.
Wenn wir Fragen beantworten wollen wie "In welchem Haus gibt es keine Vorschulkinder" oder "Wieviele Erwachsene leben in den 6 Häusern?" oder "In welchem Haus wohnen die wenigsten Kinder", dann müssen wir jedem der 24 Einträge einen Namen geben, der sich auf die Position in dieser Tabelle bezieht. Man schreibt in Fortran: INTEGER T(6,4).
Damit sind 24 Variable definiert:

T(1,1)	T(1,2)	T(1,3)	T(1,4)
T(2,1)	T(2,2)	T(2,3)	T(2,4)
T(3,1)	T(3,2)	T(3,3)	T(3,4)
T(4,1)	T(4,2)	T(4,3)	T(4,4)
T(5,1)	T(5,2)	T(5,3)	T(5,4)
T(6,1)	T(6,2)	T(6,3)	T(6,4)

Jede solche Variable nennt man ein zweifach indiziertes Feldelement.
Der 1. Index bezeichnet die Zeile, der 2. die Spalte. Allgemein ist das Element T(i,j) dasjenige in der i-ten Zeile und der j-ten Spalte.

Beispiel 5.3: Statistischer Überblick 1

```
      PROGRAM P53
* Das Programm liest die 24 Werte (4 pro Zeile), das Ergebnis
* ist die Gesamtzahl der Grund/Hauptschüler in allen Häusern.
      INTEGER T(6,4), GSumme
*  einlesen 24 Werte, 1 Zeile zu je 4 Werten auf einmal
      PRINT*,'Bitte 6 Zeilen zu je 4 Zahlen eingeben'
      DO i=1,6
        READ*, (T(i,j),j=1,4)
      ENDDO
*  Aufsummieren Grund/Hauptschüler
      GSumme = 0
      DO i=1,6
        GSumme = GSumme + T(i,2)
      ENDDO
*  Ausdruck Ergebnisse
      PRINT*,'Anzahl aller Grund/Hauptschüler:', GSumme
      END
```

Beispiel 5.4: Statistischer Überblick 2

```
      PROGRAM P54
* Gefragt ist das Haus mit den meisten Menschen.
      INTEGER T(6,4), Haus, MaxHaus, Max, Einwohner
      PRINT*,'Bitte 6 mal 4 Zahlen eingeben'
      READ*, ((T(i,j),j=1,4),i=1,6)
      Max     = 0
      MaxHaus = 0
      DO Haus = 1,6
*  Zahl alle Menschen in den Häusern
        EINWO = 0
        DO j=1,4
          Einwohner = Einwohner + T(Haus,j)
        ENDDO
*  welches ist das meistbewohnte?
        IF ( Einwohner .GT. Max ) THEN
          Max = Einwohner
          MaxHaus = Haus
        ENDIF
      ENDDO
      PRINT*,'Maximalzahl der Einwohner ist',Max,' in Haus',MaxHaus
      END
```

Wenn wir die 24 Werte in 6 Zeilen zu je 4 ausdrucken wollen, können wir schreiben:

```
      DO i=1,6
        PRINT*, (T(i,j),j=1,4)
      ENDDO
```

oder in 4 Zeilen mit 6 Werten (spaltenweise)

```
      DO j=1,4
        PRINT*, (T(i,j),i=1,6)
      ENDDO
```

Würden wir aber zum Einlesen (entsprechendes gilt auch für die Ausgabe)

```
READ*,T
```

verwenden, würden die Werte spaltenweise eingelesen, d.h. die ersten 4 Werte in Spalte 1, die nächsten 4 in Spalte 2 usw.; in diesem Fall würde nur das 1. Element T(1,1) und das letzte T(6,4) am richtigen Platz stehen. Dasselbe gilt für PRINT*,T. Man sagt auch: Fortran liest und schreibt ganze Felder ohne Angabe von Indizes spaltenweise. Dies ist bei Verwendung des nackten Feldnamens zu beachten (auch bei DATA!).

Übungen

1. Betrachten Sie die folgende Tabelle:

```
2  5  -7  3
4  8  22  7
5  9  -6  3
```

   a) Wieviele Zeilen und Spalten hat die Matrix?
   b) Welches ist das Element in Position (3,2) ?
   c) Welches sind die Positionen der Primzahlen?
   d) Was ist die Summe der Elemente der 2. Zeile, der 3. Spalte?

2. Nehmen Sie an, A sei ein INTEGER-Feld mit 2 Zeilen, 3 Spalten; B mit 3 Zeilen, 2 Spalten.
   a) Deklarieren Sie die Felder.
   b) Welcher Index kann in A auftreten und in B nicht?
   c) Nehmen Sie an, A(i,j) bekommt die Werte i+j für alle i,j und B(i,j) entsprechend i-j für alle Elemente.
   ca) Für welche Paare i,j gilt B(i,j)+A(i,j)=2*i ?
   cb) Für welche Paare i,j gilt B(i,j)-A(i,j)=-2*i ?
   cc) Welchen Wert hat A(1,1)*B(1,1)+A(1,2)*B(2,1)+A(1,3)*B(3,1) ?
   cd) Was kommt heraus mit

```
PRINT*, ((A(i,j),j=1,3),i=1,1)
PRINT*, (A(i,k)*B(k,j),k=3,5,4)
PRINT*, MOD(A(2,i),A(1,i)
```

   ce) Welchen Wert hat A(B(2,1),A(1,2) ?
   B(A(2,B(3,1))-1,B(A(1,B(3,1)),1)) ?

3. In welcher Reihenfolge erscheinen die Daten? (Q sei REAL Q(4,2))

```
a)  PRINT*, Q
b)  PRINT*, (Q(j,1),Q(j,2),j=1,4)
c)  PRINT*, ((Q(k,num),Q(k+2,num),k=1,2),num=1,2)
d)  PRINT*, (Q(5-j,2),Q(5-j,1),j=1,4)
```

4. Es sei INTEGER X(5,2),Y(2,5). Schreiben Sie 2 DO-Schleifen und eine Zuweisung, sodaß Y(i,j) den Wert von X(j,i) erhält. Drucken Sie X (5 Zeilen a 3) und Y (2 Zeilen a 5). Sie sehen: die Zeilen von X sind genau die Spalten von Y, die Spalten von X die Zeilen von Y.

Höher dimensionierte Felder

In Fortran kann ein Feld bis zu 7 Dimensionen haben; in der Praxis kommen jedoch selten mehr als 3 vor. Als Beispiel eines 3-dimensionalen Feldes stellen Sie sich vor, die Umfrage in den 6 Häusern hätte in 2 aufeinander folgenden Jahren stattgefunden. Wir schreiben dafür INTEGER T(6,4,2), hängen also an unsere alte Dimensionierung hinten als 3. Dimension eine 2 an (für die 2 Versionen der Umfrage). Schreiben Sie die beiden Beispielprogramme um, indem Sie die 2 mal 24 Werte einlesen und die Ergebnisse für beide Jahre getrennt ausgeben. Tip: Erweitern Sie die Programme,indem Sie das Innere der Programme mit

```
DO JAHR=1,2 ... ... ENDDO
```

umkleiden.
Bestimmen Sie Haus und Jahr, wo die meisten Menschen gezählt wurden.

Angabe einer unteren Indexgrenze

Im obigen Beispiel sollten 2 Umfragen, z.B. im Jahr 1986 und 1987, erstellt werden. Wir wollen jetzt die Jahreszahlen direkt als Index verwenden (anstelle von 1 und 2). Wir schreiben:

```
      INTEGER T(6,4,1986:1987)
```
Dies bedeutet, daß jeder 3. Index von 1986 an bis 1987 laufen kann.
Dieselbe Schreibweise auf die ersten beiden Indizes angewandt ergibt:
```
      INTEGER T(1:6,1:4,1986:1987)
```
was in diesen Fällen aber nur eine ausführlichere Schreibweise ist.

Allgemeine Form der Dimensionierung:

```
1 Dimension:
      Typ Feldname( Dimensionsangabe )
2 Dimensionen:
      Typ Feldname( Dimensionsangabe,Dimensionsangabe )
usw. bis 7 Dimensionen. Dabei ist
Typ  -      evtl. mit Länge,  also z.B. REAL oder REAL*8 usw.
Dimensionsangabe - Anzahl der Elemente
                   oder kleinster Index : größter Index
Im 1. Fall heißen die Feldelemente eines Feldes F
      F(1) F(2) ... F(Dimensionsangabe)
im 2. F(kleinster Ind.) F(kleinster Ind.+1) ... F(größter Ind.)
```

Zuweisen von Anfangswerten

Startwerte für Variable, mit denen das Programm beginnen soll, können mit der DATA-Anweisung zugewiesen werden. Beispiele:
```
      DATA I/4/,A/3./
```
Besonders bequem ist dies für Felder, da nicht jeder einzelne Wert aufgezählt werden muß. Soll ein Feld anfangs lauter Nullen erhalten, so genügt
```
     REAL F(100), Tabelle(100,100)
     DATA F /100*0./, Tabelle /10000*0./
```
Man beachte, daß zuerst im Programm die Typ-Deklarationen REAL, INTEGER usw. stehen müssen, falls vorhanden, und erst danach DATA. Der Typ der Anfangswerte muß mit dem Typ der entspr. Variablen übereinstimmen, ebenso muß die Zahl der angegebenen Anfangswerte genau gleich der Zahl der dimensionierten Feldelemente sein, dh. ein Feld muß vollständig mit Anfangswerten versorgt werden.

Zusammenfassung

1. Oft müssen wir Programme schreiben, in denen wir nicht von vornherein wissen, wieviele Variable wir brauchen.
2. Dieses Problem löst man mit der Deklaration von Feldern (Typ REAL, INTEGER, CHARACTER usw.). Feldelemente haben dynamisch veränderliche Namen, d.h. die Bezeichnung des Feldelements ändert sich, wenn der Index einen neuen Wert annimmt - aus X(i) wird z.B. X(1), X(2), X(3)..., wenn i die Werte 1,2,3... annimmt. Auf diese Weise können bel. viele Variable angesprochen werden.
3. Eindimensionale Felder denkt man sich in einer einfachen Liste angeordnet (Vektor), zweidimensionale in einer rechteckigen Tabelle.
4. Es genügt, die Zahl der Feldelemente als Dimensionierung anzugeben; die Numerierung läuft dann ab 1 bis zum Endwert. Oder man gibt Index-Untergrenze : Index-Obergrenze an, dabei sind negative Zahlen und Null möglich; die Numerierung geht ab Untergrenze bis Obergrenze.
5. Implizite DO-Schleifen erleichtern die Ein/Ausgabe von Feldern.
6. Anfangswerte für alle Variablen mit der DATA-Anweisung, für Felder besonders bequem mit dem Wiederholungsfaktor.
7. Die Dimensionierung von Feldern kann auch mit Variablen, statt Konstanten, als Indexgrenzen vorgenommen werden; dies ist jedoch nur in Unterprogrammen möglich und wird deshalb erst in Kap. 6 beschrieben.

Aufgaben

1. Lesen Sie 5 INTEGER-Variable aus einer Zeile und speichern Sie sie im Feld K. Auf 4 Zeilen drucken Sie die Werte, die Indizes der Werte, die positiv sind, die negativ sind, die Null sind.
2. Lesen Sie ein Feld mit 10 INTEGER-Werten ein. Drucken Sie alle Paare der Werte des Feldes aus, die als Summe 12 ergeben.
3. Lesen Sie ein Feld von 10 INTEGER-Werten, die das Alter von 10 Personen bedeuten sollen. Bestimmen Sie die beiden Personen, die im Alter am nächsten sind und die, die am weitesten entfernt sind.
4. A und B seien je INTEGER(8); lesen Sie die Werte mit impl. DOs ein. Drucken Sie die Werte, die je beidesmal in A und B sind; die in B, aber nicht in A sind; die in A, aber nicht in B sind; die entweder in A oder in B, aber nicht in beiden sind. Testen Sie mit
   A=(2 5 8 4 7 1 6 10)
   B=(9 8 4 10 3 5 12 11)
5. Erweitern Sie Aufg. 8 von Kap. 4 bzw. Beispiel 5.2 so, daß die Quotienten ebenfalls in einem Feld gespeichert werden; man drucke in 2 Kolonnen nebeneinander FiboZahl(i+1) / FiboZahl(i) und FiboZahl(i) / FiboZahl(i-1). Was fällt dabei ins Auge?
6. Lesen Sie 4 Zeilen ein, die je 7 INTEGER-Werte für die täglichen Temperaturen enthalten; berechnen Sie die durchschnittliche Temperatur, das Maximum, Mimimum pro Woche. Dasselbe für alle 4 Wochen. Die Ergebnisse sollen in einer Tabelle so aussehen:

	Durch.	Max	Min
Woche1	x	x	x
Woche2	x	x	x
Woche3	x	x	x
Woche4	x	x	x
4-Wochen-Statistik	x	x	x

7. Fügen Sie Anweisungen hinzu, sodaß Durchschnitt, Maximum und Minimum für jeden Wochentag berechnet werden. Drucken Sie die Ergebnisse in einer Tabelle aus.
8. Die folgende Aufgabe ist eher für Numeriker gedacht:
   Die Ackermann-Funktion A(i,j) ist wie folgt definiert:
   ```
   A(0,j) = j+1
   A(i,0) = A(i-1,1)
   A(i,j) = A(i-1,A(i,j-1))
   ```
   Sie dient zum Testen von rekursiven Funktionsaufrufen.
   Schreiben Sie ein Programm, das die Werte von A(0,0) bis A(10,10) in einer 11 mal 11 -Matrix speichert. Das Element der Zeile i Spalte j soll den Wert von A(i,j)) erhalten.
   Die 11 mal 11 - Matrix ist dann mit A(0:10,0:10) zu dimensionieren.
   Man beachte, daß mit diesem Verfahren anstelle des Funktionswertes an der Stelle i,j das Matrix-Element mit dem Index (i,j) tritt, also gar keine Funktion, sondern ein Matrixelement berechnet wird.
   Schwieriger ist nur herauszufinden, in welcher Reihenfolge die Berechnung erfolgen muß, damit sich alle Elemente der Reihe nach aus bereits berechneten Elementen ergeben und nicht irgendwann ein noch unbekanntes Element herangezogen wird (was beim rekursiven Funktionsaufruf ja gerade gemacht werden darf und soll).
   Tip: Die 11x11-Matrix läßt sich so nicht vollständig berechnen, allenfalls die linke obere Dreiecksmatrix einschl. Nebendiagonale (von links unten nach rechts oben), also

   ```
   ( .... )
   ( ...  )
   ( ..   )
   ( .    )
   ```

   am Beispiel einer 4x4-Matrix gezeigt.

Noch einmal: Schleifensteuerung; der Typ LOGICAL

Im Kap. 4 hatten wir bereits besprochen, wie mehrere Endebedingungen in einer Schleife geschrieben werden. Hier soll das Problem an Hand eines Beispiels mit einem Feld noch einmal systematisch vorgeführt werden.

Beispiel 5.5: Suchen in einer Liste

```
      PROGRAM P55A
* Ein bestimmter Name soll in einer vorgegebenen Liste gesucht werden
      CHARACTER*5 Liste(10), Name
      DO i=1,10
        IF (Name.EQ.Liste(i)) ???          gefunden! was nun?
      ENDDO                                wenn i > 10, geht es mit der
      PRINT*,'nichts gefunden'             folgenden Zeile weiter
      END
```

Was soll an der Stelle ??? passieren? Dazu die folgenden Überlegungen:

Der Ausgang über ENDDO wird auch "regulärer" Ausgang der Schleife genannt: die Schleife wurde solange durchlaufen, bis die eingebaute Logik "beende die Schleife, sobald i > 10" zum Tragen kommt. Dies ist der Fall, wenn die Liste erfolglos bis Ende durchsucht wurde.

Ein 2. Ausgang aus der Schleife, wenn nämlich etwas gefunden wurde, darf aber nicht auch zu der auf ENDDO folgenden Zeile führen, da dort die Meldung "nicht gefunden" steht, sondern zu einer weiter unten stehenden Zeile. Zur Lösung des Problems verwenden wird die

Methode: strukturiert mit Einbau einer logischen Variablen,
QUIT verläßt die Schleife, es geht weiter hinter ENDDO

Zunächst: Der **Typ LOGICAL** deklariert eine Variable, die nur die Werte .TRUE. (wahr) oder .FALSE. (falsch) annehmen kann. Man kann sich so z.B. das Ergebnis eines Vergleichs merken. Beispiel s.u.

```
      PROGRAM P55B
* Ein bestimmter Name soll in einer vorgegebenen Liste gesucht werden
      CHARACTER*5 Liste(10), Name
      LOGICAL gefunden
      DO i=1,10
       gefunden = Name .EQ. Liste(i)
       IF(gefunden) QUIT
      ENDDO
      IF (.NOT. gefunden) THEN
       PRINT*,'nichts gefunden'
      ELSE
       PRINT*,'gefunden als Nr.',i
      ENDIF
      END
```

Ein weiterer Vorzug der strukturierten Methode ist, daß die Einführung der logischen Variablen, die das Ergebnis des Suchvorgangs speichert, es möglich macht, an einer späteren Stelle die Information "gefunden" oder "nicht gefunden" weiterzuverwenden. Der Ablauf eines strukturierten Programms dokumentiert sich also automatisch selbst.

Die QUIT-Anweisung ist ein Vorgriff auf den Fortran90-Standard (dort heißt es EXIT mit derselben Logik).

Wir wollen das ganze zu einem sinnvollen Programm ausgestalten:

Aufgabe: Es sei ein Feld von 10 Namen vorhanden. Es soll ein bel. Name am Schirm eingelesen werden; das Programm soll die Meldung ausgeben, ob der Name in der Liste vorhanden ist oder nicht. Das ganze soll in einer Schleife bis zu 20 Mal laufen. Vorzeitiges Ende, wenn der Name nur ein Leerzeichen ist.

Algorithmus:

1. Deklariere ein Feld mit 10 Elementen und einen extra Namen
2. Gib die 10 Namen vor
3.0 Setze eine Schleife für 20 Durchgänge an
3.1 Fordere zur Eingabe eines Namens auf
3.2 Lies den neuen Namen
3.3 Test auf vorzeitiges Ende
3.4.0 Setze eine Schleife zum Durchsuchen des Feldes auf
3.4.2 Vergleiche den neuen Namen mit dem entspr. Element der Liste
3.4.3 Falls gefunden, in der logischen Variablen "gefunden" vermerken
3.4.4 Verlassen der Schleife, falls "gefunden"
3.5. Je nach "gefunden" Meldung, daß Name gefunden / nicht gefunden
4. Programmende

Fortranprogramm:

```
      PROGRAM P55
      LOGICAL gefunden
      CHARACTER *5 LISTE(10), NAME*5
      DATA LISTE /'Adam','Berta','Cecil','Denis','Emil','Fritz',
     &            'Georg','Hans','Ida','Jo'/
* das Folgende bis zu 20 Mal:
      DO i=1,20
        PRINT*,'Bitte Namen angeben in ''.....'' max. 5 Buchst.'
        PRINT*,'  '' '' beendet das Programm'
        READ*,Name
        IF (Name .EQ. ' ') THEN
          PRINT*,'Du hast jetzt schon genug? wenn es sein muß...'
          STOP
        ENDIF
* Suchvorgang
        gefunden = .false.
        DO j=1,10
          gefunden = Name .EQ. Liste(j)
          IF(gefunden) QUIT
        ENDDO
* Ende der Suche; Ergebnis?
        IF (gefunden) THEN
          PRINT*,'gefunden als Nr.',j
        ELSE
          PRINT*,'nichts gefunden'
        ENDIF
      ENDDO
      PRINT*,'Du kannst ja nie genug bekommen...'
      END
```

Kommentar zu der Zeile "gefunden = Name .EQ. Liste(j)" :

Die logische Variable gefunden bekommt den Wahrheitswert des Vergleichs Name .EQ. Liste(j)
Dies ist eine elegante Schreibweise für das umständliche

```
      IF ( Name .EQ. Liste(j) THEN
         gefunden = .TRUE.
      ELSE
         gefunden = .FALSE.
      ENDIF
```

Kapitel 6: Unterprogramme

Fragen

1. Welche Vorteile bringt das Schreiben von Fortranprogrammen in einzelnen separaten Teilen (Programm-Einheiten) ?
2. Wann werden allgemeine wann Unterprogramme (SUBROUTINE-Anweisung), wann Funktionen (FUNCTION-Anweisung) verwendet?
3. Was versteht man unter einer modularen Programmiersprache?

Programmeinheiten

Bisher hatte unser Fortran-Programm folgende Form:

```
1.        PROGRAM name
2.           Deklarationen (Typen, Felder, DATA)
3.           Anweisungen
4.        END
```

Dies nennt man eine Programmeinheit; das Fortran-Programm bestand also bisher nur aus dem Hauptprogramm.
Vom Hauptprogramm aus können weitere Unterprogramme aufgerufen werden; jedes Unterprogramm bildet wieder eine Programmeinheit; deren Form ist (da es 2 Sorten von Fortran-Unterprogrammen gibt):

```
1.  SUBROUTINE name ( Parameter-Liste )   bzw.  FUNCTION name ( .. )
2.    Deklarationen (Typen, Felder, DATA)
3.    Anweisungen
4.  END
```

also bis auf den Kopf des Unterprogramms dieselbe Struktur wie das Hauptprogramm; wichtig ist aber, daß die END-Anweisung im Unterprogramm zwar dieses beendet, man aber anschließend wieder ins Hauptprogramm zurückkehrt, nämlich hinter die Stelle des Aufrufs. Ein Unterprogramm kann auch ein zweites aufrufen usw. (aber nicht rekursiv), sodaß die Rückkehr dann jeweils ins übergeordnete (rufende) Programm geschieht.

Der Aufruf einer selbst geschriebenen Funktion geschieht wie der Aufruf einer eingebauten: durch den Funktionsnamen, dem in Klammern die Argumenten-Liste folgt. Der Funktionsname wird durch den berechneten Wert ersetzt und die Auswertung des Ausdrucks fortgesetzt.

Der Aufruf einer Subroutine geschieht mit der Anweisung

```
CALL name ( Argumenten-Liste )
```

Die Argumente werden den Parametern zugeordnet, d.h. deren Werte sind unter den Namen der Parameter verfügbar (s.u.).

Subroutine-Unterprogramme

Beispiel 6.1:

Die Fakultät von n soll als Unterprogramm (Subroutine) mit dem Namen Fak geschrieben werden:

```
     SUBROUTINE Fak(n,Fakn)
>
     IF (n.GT.1) THEN
       DO i=2,n
         Fakn = Fakn * i
       ENDDO
     ENDIF
     END
```

Die 1. Zeile enthält 3 Informationen:
1. Art des Unterprogramms (SUBROUTINE)
2. Name des Unterprogramms (FAK)
3. Parameter des Unterprogramms (N,FAKN)

Wir benutzen dies, um "k über l" zu berechnen nach der Formel
k über l = k! / (l! * (k-l)!) (! bezeichnet die Fakultät)

Beispiel 6.2: K über L

```
      PROGRAM P62
      INTEGER k,l, fakk, fakl, fakk_l
      READ*, k,l
      CALL FAK( k, fakk)
      CALL FAK( l, fakl)
      CALL FAK( k-l, fakk_l)
      PRINT*, k ' über',l, ' =', fakk / (fakl * fakk_l)
      END
```

Anmerkungen:
1. Der Aufruf einer Subroutine erfolgt mit der CALL-Anweisung:
   "CALL", dann den Namen des Unterprogramms, dann die Argumente.
2. Argument kann eine Konstante, ein Ausdruck oder eine Variable sein.
   Sie liefern Werte an die Parameter der Subroutine. Variable können
   neue Werte durch die Subroutine bekommen und diese zurückgeben.
3. Die Zuweisung eines Wertes an einen Parameter in der Subroutine, wo
   das entspr. Argument eine Konstante oder ein Ausdruck war, führt zu
   einem *ERR*. Variablen-Argumente dürfen (z.B. durch Zuweisung) im
   Unterprogramm verändert werden. Dies muß beim Aufruf beachtet werden.
4. Die END-Anweisung beendet das Unterprogramm, schreibt die evtl. ver-
   änderten Werte der Parameter an die Argumente zurück und kehrt ins
   rufende Programm zurück. Dies ist auch durch die RETURN-Anweisung
   an bel. Stelle des Unterprogramms möglich (analog der STOP-Anweisung
   im Hauptprogramm).
5. Ein Fortran-Programm kann also aus dem Hauptprogramm (gekennzeichnet
   durch die PROGRAM-Anweisung) und mehreren Unterprogrammen bestehen.
   Jede solche Programmeinheit wird mit END abgeschlossen.
   Die Reihenfolge ist nicht vorgeschrieben.

Beispiel 6.3:

```
      SUBROUTINE SUB(a,b)
      a=a+b
      END
*
      PROGRAM AplusB
      x=1.
      y=1.
      DO i=1,5
        CALL SUB(x,y)
        PRINT*,x
      ENDDO
      PRINT*,y
      END
```

Der Wert von y bleibt gleich, weil der Parameter b sich nicht ändert;
der Wert von x erhöht sich jeweils um 1. (den Wert von y).

Beispiel 6.4:

Wir schreiben eine Subroutine, um in einem Feld von 10 INTEGERs
die Nummer des kleinsten Elements zu bestimmen.
Der Algorithmus nimmt das 1. Element und vergleicht dies mit dem
folgenden; ist dies kleiner, wird dieses weiterverwendet.

```
      SUBROUTINE KLEINSTES (Feld, Nummer)
      INTEGER Feld(1:10), Test, i
      Nummer = 1
      Test = Feld(Nummer)
      DO i=2,10
        IF (Feld(i) .LT. Feld(Nummer) THEN
          Test =(Feld(i)
          Nummer = i
        ENDIF
      ENDDO
      END
```

Beispiel 6.5: Sortierprogramm

Algorithmus:
1. Wir lesen die 10 Werte in des Feld ein
2. führe folgendes 10 mal aus:
2.1 finde das kleinste Element und drucke es
2.2 sorge dafür, daß dieses Element nie wieder das kleinste wird
3. Ende

```
      PROGRAM P65
* Sortierprogramm, benötigt die Subroutine KLEINSTES oben
      INTEGER F(10),i,n
      READ*, F
      DO i=1,10
         CALL KLEINSTES ( F, n )
         PRINT*, F(n)
         F(n) = 999999
      ENDDO
      END
```

Anmerkungen:
1. Der Name des Feldes wird mit CALL als Argument übergeben.
2. Das Feld muß in der Subroutine dimensioniert werden.
3. Der Typ entsprechender Felder muß gleich sein (beide INTEGER), die Anzahl der Elemente (Dimensionierung) ist hier ebenfalls gleich, sie darf im Unterprogramm jedenfalls nicht größer sein als im rufenden Programm).

Beispiel 6.6: Quadratische Gleichung

Wir lösen die allg. quadratische Gleichung ax^2+bx+c; die Koeffizienten sollen am Schirm eingegeben werden:

```
      PROGRAM P66
* Quadratische_Gleichung
      REAL  a,b,c, n1,n2
      DO i=1,10
        READ*,a,b,c
        IF ( a .EQ. 0.0 ) STOP
        CALL QUAD(a,b,c,n1,n2)
        WRITE(6,100)  '1. Nullstelle:',n1,' 2.Nullstelle:',n2
      ENDDO
100   FORMAT(1X,A,G14.7,A,G14.7)
      END
```

```
      SUBROUTINE QUAD(x,y,z,Erste,Zweite)
      Real  x,y,z, Erste,Zweite, Diskr
      Diskr = y*y - 4.0*x*z
      IF ( Diskr .LT. 0 ) THEN
        PRINT*,' Gleichung hat keine reellen Nullstellen'
      ELSE
        Erste  = (-y +SQRT(Diskr) ) /(2.0*x)
        Zweite = (-y -SQRT(Diskr) ) /(2.0*x)
      ENDIF
      END
```

Bemerkungen:

1. Das Programm wiederholt sich, bis max. 10 verschiedenen Gleichungen bearbeitet sind: DO i=1,10
2. Sobald für a eine 0 eingetippt wird: STOP wegen "IF(a.EQ.0) STOP"; b und c müssen aber trotzdem beachtet werden, z.B. als "0/" .

Übungen

1. Beispiel 6.6 hat einen Schönheitsfehler:
   Falls die Nullstellen nicht reell sind, wird dies zwar angemerkt, jedoch werden (falsche) Werte angegeben. Verbessern Sie dies im Programm. Frage: welche Werte werden zu Unrecht ausgegeben?
2. Beschreiben Sie die Ausgabe der Paare Hauptprogramm/Subroutine.
   Geben Sie jeweils 3 Punkte an:
   1) Anzahl und Typ der einzelnen Argumente (Parameter)
   2) welche Werte hineingehen, welche herauskommen (Ergebnisse)
   3) was die Ergebnisse aus den eingehenden Werten machen
   NB: Das Wie (der Algorithmus) interessiert an dieser Stelle nicht!

a)
```
      PROGRAM A
        I=1
        CALL FCN(I,J)
        PRINT*,I,J
      END
```
```
      SUBROUTINE FCN(K,L)
        L=K**2-3*K+5
      END
```

b)
```
      PROGRAM B
        REAL AB(10)/10*1./
        DO N=1,4
           CALL ADD(AB)
        ENDDO
        PRINT*,AB
      END
```
```
      SUBROUTINE ADD(TAB)
        REAL TAB(10)
       DO I=2,6,2
          TAB(I-1)=TAB(I-1)-1
          TAB(I)=TAB(I)+1
        ENDDO
      END
```

c)
```
      PROGRAM C
        M=0
        DO J=1,4
           CALL R(J,M)
        ENDDO
        PRINT*,M
      END
```
```
      SUBROUTINE R(J,N)

       DO K=1,J
          N=N+K
        ENDDO

      END
```

d)
```
      PROGRAM D
        INTEGER L/1/,K/2/
        CALL HELP(I,K)
        CALL HELP(K,L)
        PRINT*,L,K
      END
```
```
      SUBROUTINE HELP(K,L)
        INTEGER KL/5/
        KL=KL*K/L
        K=KL*L

      END
```

Hinweis: beim 2. Aufruf von HELP hat KL nicht den Wert 5!

e)
```
      PROGRAM E
        KA=1
    3   CALL SUB
        KA=KA+1
        GOTO 3
      END
```
```
      SUBROUTINE SUB
        INTEGER N/1/
        IF (N.GT.10) STOP
        N=N+1

      END
```

3. Wo liegt der Fehler?

```
  a) CALL SUB(X,Y)              SUBROUTINE SUB(Z)
  b) INTEGER MAX(1)
     CALL SUB(MAX,A)            SUBROUTINE SUB(J,R)
  c) REAL A(10),J0              SUBROUTINE SUB(B,ZY)
     CALL SUB(J0,A)             REAL B(100)
  d) CALL SUB(3)                SUBROUTINE SUB(OH)
  e) CALL (A,B,C,D,E)           SUBROUTINE (W,X,Y,Z,E)
  f) CALL SUB(MAT(10),F)        SUBROUTINE SUB(MAT(10),F)
  g) CALL SUB(5.3)              SUBROUTINE SUB(S)
                                REAL S(1)
  h) X=SUB(Y)                   SUBROUTINE SUB(Y)
  i) CALL SUB(A)                SUBROUTINE SUB(1.375)
  j) ABC=5.                     SUBROUTINE SUB(RST,XYZ)
     CALL SUB(ABC,DEF)          RST=RST+XYZ
     END                        END
  k) X=2.                       SUBROUTINE SUB(X,Y,Z)
     Y=2.                       Z=X**Y
     CALL SUB(X,Y,Z)            END
     END
```

Function-Unterprogramme

Die 2. Art von Unterprogrammen ist die Funktion (FUNCTION). Sie bringt mit dem Funktionsnamen den Funktionswert zurück. Daher die Regel:

In Funktionen sollen die Parameter niemals Werte zurückbringen!

Oder anders ausgedrückt: Funktionen sollen unter keinen Umständen ihre Parameter (und damit die entspr. Argumente) verändern.
Wenn dies erforderlich ist, verwende man eine Subroutine.

Beispiel 6.7: Fakultät als Function:

```
      INTEGER FUNCTION Fak(n)
      Fak=1
      IF (n.GT.1) THEN
        DO i=2,n
          Fak = Fak * i
        ENDDO
      ENDIF
      END
```

Anmerkung:
Es gelten dieselben Regeln wie bei der Subroutine. Zusätzlich:
1. Vor "FUNCTION" kann der Typ des Funktionswertes angegeben werden (falls nicht nach der "1. Buchstaben Regel").
2. Der Funktionswert wird dem Namen der Funktion wie einer Variablen in der Function zugewiesen.
3. Der Funktionsaufruf geschieht innerhalb eines Ausdrucks.

Beispiel 6.8: K über L Version 2

```
PROGRAM P68
* benötigt die Function Fak oben
      INTEGER k,l, Fakk, Fakl, Fakk_l ,Fak
      READ*, k,l
      Fakk = Fak(k)
      Fakl = Fak(l)
      Fakk_l = Fak( k-l )
      PRINT*, k ' über',l, ' =', Fakk / (Fakl * Fakk_l)
      END
```

NB: Fak als INTEGER FUNCTION muß als INTEGER deklariert werden, da sonst nach der 1. Buchstaben Regel in P68 Fak REAL wäre!

Beispiel 6.9: K über L Version 3
---

wie P68, aber kürzer:

```
      PROGRAM P69
      INTEGER Fak
      READ*, k,l
      PRINT*, k ' über',l, ' =', Fak(k) / (Fak(l) * Fak(k-l) )
      END
```

Übung

Schreiben Sie das obige Sortierprogramm um, indem Sie statt einer Subroutine eine FUNCTION KLEINSTES benutzen.

Ein weiteres Beispiel

Beispiel 6.10

Aufgabenstellung:

Man lese 3 Zahlen ein; sie sollen die Längen der 3 Seiten eine Dreiecks sein. Wenn dies überprüft ist, soll herausgefunden werden, ob das Dreieck stumpf-, recht- oder spitzwinklig ist.
Für die Plausibilitätsprüfung müssen wir einmal ausschließen, daß einer der 3 Kandidaten für Seitenlängen kleiner 0 ist. Außerdem darf eine Seite nicht länger als die Summe der beiden anderen sein.
Sodann sagt Pythagoras, daß im rechtwinkligen Dreick die Summe der Quadrate der beiden kürzeren Seiten gleich dem Quadrat der längeren Seite ist. Ist das Quadrat größer als die Summe, ist es ein stumpfwinkliges, ist es kleiner, ein spitzwinkliges Dreieck.

Algorithmus:

```
1. Lies die 3 Werte ein.
2. Falls einer der 3 Werte < 0
      ist die Funktion Ist_ein_Dreieck FALSE (kein Dreieck)
2.2 sonst
    falls größte Seite > Summe der anderen
      ist die Funktion Ist_ein_Dreieck FALSE (kein Dreieck)
    sonst
      ist die Funktion Ist_ein_Dreieck TRUE  (Dreieck)
                in der Form: Ist_ein_Dreieck bekommt den Wahrheitswert
                   des Vergleichs "größte Seite > Summe der anderen"
   Zur Bestimmung der größeren Seite verwenden wir die Funktion MAX.
   Falls also ein Dreieck möglich ist, testen wir den Typ des Dreiecks:
3. Bestimme die längste und die kürzeren Seiten.
4. Berechne den Wert
   längste Seite ** 2  - ( mittlere ** 2 + kleinere ** 2 )
5. Falls der Wert
      < 0 : Typ ist spitzwinklig
      = 0 : Typ ist rechtwinklig
      > 0 : Typ ist stumpfwinklig
```

Wir wollen also die folgenden Funktionen verwenden:

```
      LOGICAL FUNCTION Ist_ein_Dreieck(s1,s2,s3)
```

mit der Bedeutung TRUE: ist ein Dreieck; FALSE: ist kein Dreieck.
Parameter sind die 3 Seitenlängen

```
      INTEGER FUNCTION Typ_des_Dreiecks(s1,s2,s3)
```

s1,s2,s3 wie oben; die Zahlen 1,2,3 sollen den Typ des Dreiecks angeben mit der Bedeutung 1=spitzwinklig, 2=rechtwinklig, 3=stumpfwinklig.

```
      PROGRAM P6_10
* liest 3 Zahlen (Längen) und bestimmt, ob überhaupt ein Dreieck
* vorliegt; wenn ja, wird der Typ des Dreiecks bestimmt
      INTEGER spitzwinklig, rechtwinklig, stumpfwinklig
      PARAMETER ( spitzwinklig=1, rechtwinklig=2, stumpfwinklig=3 )
      REAL      Seite1, Seite2, Seite3
      LOGICAL   Ist_ein_Dreieck
      INTEGER   Typ_des_Dreiecks
* es folgen die ausführbaren Anweisungen
      WHILE( .TRUE. )
      PRINT*,'Bitte die 3 Seiten eingeben; Strg_z endet!'
      READ(*,*,END=99) Seite1,Seite2,Seite3
      PRINT*,'Die Längen sind:'
      PRINT*,Seite1,Seite2,Seite3
      IF( Ist_ein_Dreieck(Seite1,Seite2,Seite3) )THEN
         PRINT*, '    Das Dreieck ist '
         SELECT( Typ_des_Dreiecks ( Seite1,Seite2,Seite3 ) )FROM
           CASE spitzwinklig
                PRINT*,'    spitzwinklig'
           CASE rechtwinklig
                PRINT*,'    rechtwinklig'
           CASE stumpfwinklig
                PRINT*,'    stumpfwinklig'
         ENDSELECT
      ELSE
         PRINT*,'   Das gibt kein Dreieck'
      ENDIF
      ENDWHILE
  99  PRINT*,'Programmende'
      END
 *
      LOGICAL FUNCTION Ist_ein_Dreieck( s1,s2,s3 )
      REAL  s1, s2, s3, Laengste_Seite
      IF ( s1 .LE. 0  .OR.  s2 .LE. 0  .OR.  s3 .LE. 0 ) THEN
           Ist_ein_Dreieck = .FALSE.
      ELSE
           Laengste_Seite = MAX( s1, s2, s3)
           Ist_ein_Dreieck = Laengste_Seite .LT. 0.5 * (s1 + s2 + s3)
 *         je nachdem wird Ist_ein_Dreieck TRUE oder FALSE
      ENDIF
      END
 *
      INTEGER FUNCTION Typ_des_Dreiecks(s1,s2,s3 )
      INTEGER spitzwinklig, rechtwinklig, stumpfwinklig
      PARAMETER ( spitzwinklig=1, rechtwinklig=2, stumpfwinklig=3 )
      REAL      s1,s2,s3   ,wenig
      PARAMETER ( wenig = 0.000001 )
      REAL      lang, mittel, kurz, Differenz
      lang   = MAX (S1, S2, S3)
      kurz   = MIN (S1, S2, S3)
      mittel = S1 + S2 + S3 - ( lang + kurz )
      Differenz = lang**2 - ( mittel**2 + kurz**2 )
      IF( ABS(Differenz) .LT. wenig )THEN
          Typ_des_Dreiecks = rechtwinklig
      ELSEIF( Differenz .GT. 0 )THEN
          Typ_des_Dreiecks = stumpfwinklig
      ELSE
          Typ_des_Dreiecks = spitzwinklig
      ENDIF
      END
```

Als Dateneingabe nehmen Sie z.B.

```
  3  4  5
  3  4  6
  3  4  4
  3  4 10
  3  4 -1
  0  1  1
```

und dann Strg_z (am PC)

Kommentar:

1. Die PARAMETER-Anweisung gibt den in der Klammer angegebenen Namen festen Werte. Man nennt dies "Deklaration von Konstanten".
2. Die Schleife WHILE(.TRUE.) ... ENDWHILE ist eine Endlosschleife. Sie wird nur über die Aktivierung von "END=99" verlassen.
3. Die SELECT-Anweisung SELECT(integer-wert)FROM
   CASE integer-wert oder von:bis oder ..,..,..
   ENDSELECT
   stellt einen Vorgriff auf den Fortran90-Standard dar (dort heißt es SELECT CASE(integer-wert) ... ENDSELECT
   So können in Abhängigkeit des INTEGER-Wertes eine Liste von Aktionen ausgeführt werden. Ohne diese Anweisung müßte man schreiben:
   IF(integer-wert .EQ. 1) THEN ...
   ELSEIF(integer-wert .EQ. 2) THEN ...
   ELSEIF(integer-wert .EQ. 3) THEN ...
   ENDIF

Module, Programmbibliotheken

Ein Modul ist eine Unterprogrammeinheit, die mindestens die folgende Eigenschaft aufweist:

> Eine modulare Unterprogrammeinheit bekommt ihre Information von der rufenden Programmeinheit **nur** über die Parameter-Liste;
> sie liefert Information **nur** über diese (bei Funktionen über den Funktionsnamen) an das rufende Programm zurück

Daraus ergibt sich als Konsequenz:

1. Die Namensgebung der Variablen, Felder usw. ist für jede modulare Programmeinheit beliebig; der Programmierer muß nur die Übersicht über die Namen der laufenden Programmeinheit behalten.
   Diese Größen nennt man "lokal", s.u. "Gültigkeitsbereich von Namen".
2. Unterprogramme können von verschiedenen Programmierern geschrieben werden, wenn eine Übereinkunft über die Art der Parameter (und damit der Argumente) besteht: die "Schnittstelle" zwischen den Moduln muß bekannt sein ("öffentlicher" Teil des Unterprogramms); alles andere, also auch die Methode ("privater" Teil des Unterprogramms), kann dem rufenden Programm, kann dem Anwender unbekannt bleiben.
3. In Fortran wurde damit das Bibliothekskonzept verwirklicht: man kann selbst geschriebene oder von anderen erstellte Unterprogramme (Unterprogrammbibliotheken) benutzen, aus denen der Fortran-Compiler automatisch die jeweils benötigten (aufgerufenen) Unterprogramme (Subroutine oder Function) auswählt und zum Laufen bringt.

Aufgaben

1. Schreiben Sie eine Subroutine, deren Parameter ein Vektor mit 10 INTEGERs ist. Das Unterprogramm soll die 10 Werte in umgekehrter Reihenfolge zurückgeben.
2. Schreiben Sie eine Function DET, die die Determinante einer 2x2-Matrix zurückgeben soll. Die Function soll als Parameter den Namen der Matrix haben. (Formel: falls die 4 Matrixelemente a,b,c,d sind, ist deren Determinante = a*c - b*d)

3. Schreiben Sie eine Subroutine zur Lösung von 2 linearen Gleichungen mit 2 Unbekannten nach der Cramerschen Regel. Die Subroutine soll folgende Parameter haben: KOEFF (2x2-Matrix der Koeffizienten), RS (Vektor der rechten Seite), LOESUNG (Vektor der Lösungen), IFLAG bekommt den Wert 0, falls die Gleichungen keine Lösung haben, 1 wenn nur 1 Lösung, 2 wenn es unendlich viele Lösungen gibt;
   z.B. für 2x + 3y = 12   ist KOEFF= (2. 3.), RS= (12.)
            -4x + 8y = 4   (-4. 8.)   (4.)
   Das Ergebnis wäre dann LOESUNG = (3. 2.) und IFLAG = 1
   Schreiben Sie ein Hauptprogramm, welches diese Werte mit DATA bereitstellt, die Subroutine aufruft und IFLAG auswertet.

4. Schreiben Sie eine Subroutine namens RANDI zur Erzeugung von 60 "Zufalls"-Zahlen INTEGER R(60) im Bereich j bis k. Methode:
   Setze R(1) = eine positive INTEGER-Zahl < 65536. Für N=2,3,...:
   R(n) = MOD(25173*(R(n-1),65536). Dies erzeugt 6 Zahlen im Bereich 1...65535. Um auf den Bereich j...k zu kommen, ersetze man R(i) durch R(i)/65535.0 *(k-j+1) + j
   Die Parameter von RANDI sind R,J und K.
   Testen Sie Ihr Unterprogramm durch ein Hauptprogramm, welches die 60 Zufallszahlen in 10 Zeilen zu je 6 Werten ausdruckt.

5. Verwenden Sie RANDI für das folgende Problem:
   Erzeugen Sie 200 Zufallszahlen im Bereich 1...6. Nehmen Sie jeweils 2 Zahlen zusammen zu einem Wurf zweier Würfel.
   Schreiben Sie ein Programm, welches folgende Frage beantwortet:
   Wie oft traten die 36 möglichen Wurfergebnisse auf?
   Schreiben Sie die Ergebnisse in eine 6x6-Tafel, wo die Zeile den Wert des 1. Würfels, die Spalte den des 2. Würfels in jedem Doppelwurf bedeuten. Mit welcher relativen Häufigkeit traten die 11 möglichen Summen 2,3,...,12 auf? Drucken Sie entsprechende Header, gefolgt von 11 Zeilen Statistik.

6. Schreiben Sie eine Subroutine mit den 2 Parametern: K, ein Feld von 20 INTEGERs, und INSERT. Die Werte von K und von INSERT werden an das Unterprogramm übergeben. Der Wert von INSERT soll nichtabsteigend (d.h. aufsteigend oder gleichgroß) einsortiert werden.
   Schreiben Sie ein Hauptprogramm,das die 20 Werte auf 0 initialisiert und dann 20 mal je 1 Wert einliest und einsortiert. Drucken Sie am Ende die 20 Werte aus (sie müssen steigende Werte haben).

7. Schreiben Sie ein Programm, welches das Pascalsche Dreieck max. 10. Ordnung ausgibt. Definition der Zeile n im Pascalschen Dreieck:
   (n)   (n)   (n)   (n)
   (0) , (1) , ... , (n)   wo (m) "n über m" bedeuten soll.
   Programmende bei Eingabe einer negativen Ordnung.
   Man verwende folgende Hauptprogrammstruktur:

```
      INTEGER Zeile(11), ...
      WHILE ( .TRUE. )
         CALL EINGABE ( Ordnung, Max_Ordnung, Ende )
         IF ( Ende ) QUIT
         CALL BERECHNE ( Zeile, Ordnung )
      ENDWHILE
      PRINT*,...
```

   sowie die folgenden Unterprogramme:

```
      SUBROUTINE EINGABE ( ... )
      SUBROUTINE BERECHNE ( ... )
*     ruft UEBER auf
        CALL AUSGABE ( Zeile, Nummer )
      SUBROUTINE AUSGABE ( ... )
      INTEGER FUNCTION UEBER ( a,b )
*     ruft FAK auf
      INTEGER FUNCTION FAK ( n )
```

Dimensionierung zur Ausführungszeit

In allen bisherigen Beispielen war die Dimensionierung der Felder im Unterprogramm gleich der im übergeordneten Programm. Dies ist jedoch nicht erforderlich, wenn man die Regel beachtet, daß keine Dimension durch ein Unterprogramm erweitert werden kann. Man erhält damit eine Möglichkeit, mit demselben Unterprogramm Vektoren und Matrizen mit unterschiedlicher Größe (unterschiedlicher Anzahl der Elemente in jeder Dimension) verarbeiten zu können.

Beispiel 6.11

```
   PROGRAM HAUPT           SUBROUTINE SUB(X,N)     SUBROUTINE SUB(X)
   REAL A(20), B(100)      REAL X(n)               REAL X(*)
   CALL SUB(A,20)          DO i=1,n...
   CALL SUB(B,100)            ......
   CALL SUB(B(50),30)            ......
   END                     ENDDO                   ENDDO
                           END                     END
```

Hier sehen wir, daß dieselbe Aktion von SUB einmal mit den 20 Elementen von A und dann mit den 100 Elementen von B durchgeführt wird. Der Wert N in der Dimensionierung von X besagt, daß mit X einmal die 20 Elemente von A und dann die 100 von B angesprochen werden. Im 3. Fall hat X die 30 Elemente von B ab dem 50. (ab B(50)), also B(50),B(51),...B(79).

Es geht jedoch auch kürzer:

```
     PROGRAM HAUPT           SUBROUTINE SUB(X)
     REAL A(20), B(100)      REAL X(*)
     CALL SUB(A)             DO i=1,n...
     CALL SUB(B)                ......
     CALL SUB(B(50))               ......
     END                     ENDDO
                             END
```

Der Stern in der Dimensionierung von SUB übernimmt automatisch die Dimensionierung aus dem rufenden Programm.

Beispiel 6.12

Die Matrix (1 4 7) wird bearbeitet (denken Sie daran, daß Fortran
(2 5 8)
(3 6 9)
eine Matrix spaltenweise mit Werten füllt!)

```
    PROGRAM merkwuerdig                  SUBROUTINE SUB(X,n,m)
    INTEGER A(1:3,1:3)/1,2,3,4,5,6,7,8,9/   INTEGER X(1:n,1:m)
    CALL SUB(A,2,2)                      PRINT*,X
    END                                  END
```

N und M sind 2, also ist X(1:2,1:2) dimensioniert. Es sollte also der obere linke 2x2-Ausschnitt der Matrix A mit PRINT*,X ausgegeben werden, also (1 4) bzw. in der Fortran-Reihenfolge (spaltenweise) die Zahlen
(2 5)
in der Reihenfolge 1 2 4 5.
Das Ergebnis ist aber 1 2 3 4. Warum?
Die Deklaration A(1:3,1:3) reserviert 9 Speicherplätze für die 9 Elemente, spaltenweise und hintereinander. X(1:2,1:2) tut dasselbe mit 4 Elementen, wobei das 1. Element von X mit dem 1. Element von A identisch sind. Wir haben also folgende Sicht der Dinge:
Das Hauptprogramm sieht den Speicher so:

< Spalte 1 von A >			**< Spalte 2 von A >**			**< Spalte 3 von A >**		
1	2	3	4	5	6	7	8	9

Das Unterprogramm sieht den Speicher wie unten ...
Wie können wir dann aber die erwünschte 2x2- Teilmatrix von A an das Unterprogramm übergeben und ausdrucken? Wir übergeben das Feld A mit der Originaldimensionierung und benutzen die beiden Werte 2,2 zur Markierung des entspr. Ausschnitts der Matrix:

```
PROGRAM teilmatrix            SUBROUTINE SUB(X,Norig,Morig,n,m)
INTEGER A(1:3,1:3)/.../       INTEGER X(Norig,Morig)
CALL SUB(A,3,3,2,2)           PRINT*, (( X(i,j), i=1,n), j=1,m)
END                           END
```

Auch hier können wir die rechts stehende (letzte) Dimensionierung mit dem Stern automatisch übernehmen mit INTEGER X(NORIG,*).
Der Aufruf und die Parameter-Liste von SUB verkürzen sich entsprechend:
CALL SUB(A,3,2,2) mit SUBROUTINE SUB(X,NORIG,N,M)
Leider funktioniert der Stern nicht mit den weiter vorn stehenden Dimensionen, INTEGER X(*,*) ist also nicht möglich.

Beispiel 6.13

Es werden 20 Elemente in ein REAL-Feld eingelesen.
Es sollen die Durchschnittswerte mit der Function MITTEL berechnet werden, und zwar von allen 20, von allen außer dem 1., dem 2. usw. bis zum letzten Element.

```
PROGRAM gleitender_Durchschnitt   REAL FUNCTION Mittel(X,n)
REAL Val(20),Mittel               REAL X(*)
READ*,Val                         Mittel = 0.
DO i=1,20                         DO i=1,n
   PRINT*,Mittel(Val(i),21-i)        Mittel=Mittel+X(i)
ENDDO                             ENDDO
END                               Mittel=Mittel/n
                                  END
```

Der 2. Parameter von MITTEL gibt die Zahl der Elemente von X an, die in der DO-Schleife aufaddiert werden sollen. Für I=1 im Hauptprogramm sollen alle 20 genommen werden, also steht im Aufruf als 2. Parameter 21-I. NB: Jeder andere kleinere (positive) Wert wäre auch zulässig. Man beachte auch, daß MITTEL im rufenden Programm als REAL deklariert werden muß, da sonst die Erste-Buchstaben-Regel MITTEL als INTEGER betrachten würde.

Externe Unterprogramme

Wir wollen ein Unterprogramm schreiben, welches das Minimum einer ganz beliebigen Funktion bestimmen soll. Nehmen wir an, wir hätten die beiden Funktionen

FA(i) = $(50-i)^2/i^2$ für i=1,2,...,100
FB(i) = $120 - (i-60)^2$ für i=1,2,...,100

und folgendes Programm:

Beispiel 6.14

```
PROGRAM P6_14                     SUBROUTINE MINI(klein,F)
EXTERNAL FA,FB                    klein=F(1)
CALL MINI(m,FA)                   DO i=2,100
PRINT*,'Minimum von FA ist',m       IF ( F(i) .LT. klein) klein=F(i)
CALL MINI(m,FB)                   ENDDO
PRINT*,'Minimum von FB ist',m     END
END
FUNCTION FA(j)                    FUNCTION FB(k)
FA=(50-j)**2/j**2                 FB=120-(k-60)**2
END                               END
```

Das 2. Argument von "CALL MINI" übergibt nicht den Wert einer Funktion, sondern den Namen, so dieser in einer EXTERNAL-Deklaration des rufenden Programms enthalten ist. Das gerufene Unterprogramm hat den Platzhalter "F" für den jeweils übergebenen Funktionsnamen bereit und ruft beim 1. Aufruf FA, beim 2. FB. Selbstverständlich muß dem Unterprogramm MINI Typ und Zahl der Argumente der Funktionen bekannt sein: alle mit "F" gerufenen Funktionen müssen offenbar dieselbe Parameter-Liste haben (Typ und Anzahl), können aber mit beliebigen Werten aufgerufen werden.

Zusammenfassung: Wirkungsweise des Aufrufs

Wir schreiben:

```
         CALL abc( Argumenten-Liste )       mit der Subroutine abc:
   SUBROUTINE abc( Parameter-Liste )
oder
      var = xyz( Argumenten-Liste )         mit der Function xyz:
   FUNCTION xyz( Parameter-Liste )
```

Die Elemente der Argumenten-Liste werden der Reihe nach den Elementen der Parameter-Liste zugeordnet; sie müssen in Anzahl, Typ und Länge (*4, *8 usw.) übereinstimmen.

Für die Übergabe von Werten (Normalfall) gilt:

Die Werte der Argumente bekommen innerhalb des gerufenen Unterprogramms die Namen der entsprechenden Parameter. Beim Beendigen (RETURN) des Unterprogramms erfolgt die umgekehrte Zuordnung, d.h. die Werte der Parameter (die durch das Unterprogramm verändert werden können) haben im rufenden Programm wieder die Namen der Argumente.

Achtung: In einer Function sollen die Werte der Parameter nicht verändert werden. Nur der Funktionswert soll einen Wert liefern.

Die Feldelemente der Argumenten-Liste bzw. das 1. Element eines Feldes, falls dieses angegeben ist, werden dem entsprechenden Feldelement bzw. dem 1. Element eines Feldes des Unterprogramms zugeordnet. Dabei ist zu beachten, daß das gerufene Feld insgesamt (gemäß seiner Deklarierung) nicht außerhalb des übergebenen Feldes zu liegen kommt:

Ein Unterprogramm darf ein übergebenes Feld durch seine Dimensionierung nicht vergrößern (nach oben oder unten)

Einer Konstanten, einem Ausdruck, einer DO-Schleifen-Steuergröße darf kein Parameter entsprechen, dessen Wert im Unterprogramm geändert wird.

Im Funktions-Unterprogramms muß dem Funktionsnamen (als Variable betrachtet) der gewünschte Funktionswert zugewiesen werden. Nach Beendigung der Funktion wird dieser Wert als Funktionswert zurückgebracht.

Für die Übergabe von Namen von Unterprogrammen gilt:

Diese werden mit "EXTERNAL UProg-Liste" im rufenden Programm deklariert. Diese Unterprogramme bekommen innerhalb des gerufenen Unterprogramms die Namen der entsprechenden Parameter und werden dort mit diesen Namen aufgerufen. Alle externen Unterprogramme, die zum selben Parameter gehören, müssen dieselbe Parameter-Liste (Anzahl, Typ, Länge) besitzen.

Gültigkeitsbereich von Namen

Die in einem Fortran-Programm (Hauptprogramm plus evtl. Unterprogramme) enthaltenen Namen lassen sich einteilen in lokale Namen / globale Namen:

lokal sind:
- einfache Variable
- Feldvariable, Feldelemente
- Anweisungsnummern, Formatnummern

global sind:
- Standard-Funktions-Namen (Namen der eingebauten Funktionen)
- Unterprogramm-Namen (Function-Namen, Subroutine-Namen)

Lokale Namen sind nur innerhalb der eigenen Programmeinheit bekannt; außerhalb derselben bezeichnet jeder Name (auch ein gleichlautender) ein anderes Objekt (einen anderen Speicherplatz).

Globale Namen sind in allen Programmeinheiten bekannt (können von allen Programmeinheiten aus aufgerufen werden).

Mischform: Der Funktions-Name ist global (von allen Programmeinheiten aus aufrufbar), aber auch gleichzeitig lokal, denn er wird bei der Zuweisung des Funktionswertes wie eine einfache Variable behandelt.

Aufgaben (Fortsetzung von S. 58)

8. Schreiben Sie eine Subroutine, die m*n INTEGER-Werte in einer Matrix speichert. Man setze n und m als maximal 10 voraus. Das Hauptprogramm liest zunächst die aktuellen Werte n,m; sodann n*m weitere Werte. Die ersten n Werte sollen in die 1. Zeile, die zweiten in die 2. Zeile usw. der Matrix gestellt werden. Die Subroutine hat folgende 6 Parameter: die Dimensionen der Matrix wie im rufenden Programm; die aktuellen Dimensionen n,m; der Vektor für die n*m Werte; die Matrix, in die die Werte gestellt werden.

9. Schreiben Sie eine Subroutine MERGE zum Mergen von 2 aufsteigend sortierten INTEGER-Mengen. Deren Zahl der Werte soll der Subroutine übergeben werden. Die Subroutine soll ein Feld aufbauen, das die Menge beider Mengen (aufsteigend) enthalten soll. Verfahren: Man betrachte der Reihe nach jeweils ein Element der einen und der anderen Menge und lege das kleinste von beiden in die Vereinigungsmenge ab. Im Hauptprogramm seien n,m maximal gleich 10.

Kapitel 7: Ein- und Ausgabe

Fragen

1. 3 grundlegende Fragen, die immer bei Ein- und Ausgabe auftauchen:
   a) Von welchem E/A-Gerät werden die Daten gelesen bzw. auf welches werden sie geschrieben?
   b) In welcher Form sollen die Daten ein- bzw. ausgegeben werden?
   c) Welche Variablen bekommen bzw. liefern einen Wert?
2. Was bedeutet "formatfreie" Ein- und Ausgabe, was "formatierte" Ein- und Ausgabe?

Eingabe

Die Antwort auf die 3 Fragen oben:
zu a):
Eingabe-Geräte können sein:

Kartenleser, (Fernschreiber-) Lochstreifen (veraltet)
Bildschirm (Tastatur), Magnetband, Magnetkassette, Magnetplatte (größere Rechner)
Bildschirm / Fernseher (Tastatur), Diskette, Festplatte, Maus (typisch für Microcomputer)

Im Kurs haben wir den Bildschirm mit Tastatur zur Verfügung. Mit entspr. Fortran-Anweisungen (OPEN, s.u.) können wir die auf den genannten Geräten eingerichteten Datenbestände (Files auf Diskette bzw. Festplatte) mit dem Fortranprogramm verbinden. Ohne besondere Anweisungen nimmt Fortran an, daß die Daten vom Bildschirm kommen; dies nennt man "Standard-Eingabe"; sie hat in Fortran als Geräte-Nummer 5 oder einen * (s. u.).

zu b):
Was wir eintippen und als Daten einlesen, sind die Zeichen der Tastatur; das Fortranprogramm muß z.B. entscheiden, ob eine getippte Ziffer Teil einer Zahl oder Teil eines Wortes sein soll; muß Dezimalpunkte und Exponenten berücksichtigen und die einzelnen Elemente auseinanderhalten. Wie das geschehen soll, wird im Programm definiert (FORMAT) oder kann Teil der Daten selbst sein. Ersteres heißt "formatierte Eingabe", letzteres "formatfreie Eingabe".

zu c):
Dies wird ausschließlich durch das Programm bestimmt, nämlich durch die Liste der Variablen in der READ-Anweisung.

Formatfreie Eingabe haben wir bisher mit der Anweisung "READ*,..." kennengelernt. Dabei spielte der genaue Ort (Spalte), wo die Daten geschrieben waren, keine Rolle, nur die Reihenfolge.
Bei der formatierten Eingabe (FORMAT-Anweisung; kann auch Teil der READ-Anweisung sein, s.u.) wird genau festgelegt, in welche Spalten welcher Zeile das Eingabe-Datenelement geschrieben werden muß.
Außerdem können wir angeben, was geschehen soll, wenn ein READ keine Daten, überhaupt keine oder keine mehr, findet.

Formen der READ-Anweisung

Im folgenden kommt die Eingabe vom Bildschirm (Standard-Einheit 5):

formatfrei	**formatiert**
READ*, E/A-Liste	READ(*,n) E/A-Liste n FORMAT(Formatinfo)

dass. mit expliziter Geräte-Angabe * bzw. 5:

```
  READ(*,*) E/A-Liste              READ(UNIT=5,FMT=n) E/A-Liste
                                 n FORMAT( Formatinfo )
```

zusätzl. Angabe einer Anweisungsnummer für "Datenende":

```
  READ(*,*,END=e) Variablen-Liste  READ(UNIT=5,FMT=n,END=e) Var.-Liste
e ...                            n FORMAT( Formatinfo )
                                       e ...
```

Die E/A-Liste ist eine Variablen-Liste; sie besteht aus durch Komma getrennten Variablen, oder auch aus impliziten DO-Schleifen, s. Kap. 4. Der erste * im formatfreien READ bedeutet soviel wie "Standard-Einheit" d.h. Einheit 5 (Bildschirm); der 2. Stern bedeutet "formatfrei".

Formatfreie Eingabe

Die Daten werden durch Komma, Leerzeichen oder Zeilenende voneinander getrennt. Sie werden der Reihe nach den Variablen der Eingabe-Liste zugewiesen. Ein READ liest immer mind. eine neue Zeile ein; wenn diese zu wenige Daten enthält, werden automatisch weitere Zeilen eingelesen.

Wiederholungsfaktor:
Anstelle eines einzelnen Wertes kann auch w*wert stehen, wo w eine ganze Zahl >= 1 ist. Dies ist gleichbedeutend mit dem Niederschreiben von w durch Leerzeichen oder Komma getrennten identischen Werten.
Beispiel: READ*, (A(I),I=1,100) mit den Daten 100*0.0
füllt das Feld A mit 0.0

Nullelement (Leerelement):
Zwei aufeinanderfolgende Kommas stellen ein Nullelement dar. Die diesem entsprechende Variable bekommt keinen neuen Wert (behält ihren alten).
Beispiel: READ*, A,B,C mit den Daten
1,,3
gibt A den Wert 1, C den Wert 3, B bleibt ungeändert

Nullelement mit Wiederholungsfaktor:
Beispiel: READ*, (A(I),I=1,100) mit den Daten
99*, 0.0
führt dazu, daß das Feld A ungeändert bleibt bis auf A(100)=0.0

Datenende:
/ beendet Dateneingabe und READ-Anweisung; alle noch anstehenden Eingabe-Variablen behalten ihren alten Wert (Wirkung wie die Angabe von Nullelementen für die restlichen Variablen).
NB: Am PC-Schirm führt Strg_z zu End-Of-File (END=e wird ausgeführt). Fehlt END=... , so resultiert ein Error (abzufangen mit ERR=r)

Einlesen einer unbekannten Anzahl von Daten

Dazu gibt es 3 Methoden:
1. Als erstes kommt die Anzahl der noch folgenden Eingabe-Datenelemente; die Werte der Daten-Elemente können beliebig sein.
2. Eine bestimmte Zahl (ein bestimmtes Wort) gilt als letztes Element, z.B. die Zahl 999999 bei numerischer Eingabe, 'Ende' bei Texteingabe. Hier kann dieser eine Wert (dieses Wort) nicht Datenelement sein.
3. Das READ liest alles, was da ist, und merkt selbst, wenn die Daten zu Ende sind (keine Beschränkung für die Werte der Datenelemente).

Die Datenelemente werden erst einmal in einem Feld abgespeichert, bevor die Verarbeitung beginnt; diese wird durch die CONTINUE-Anweisung unten angedeutet. Alle Fehlermöglichkeiten sollen eine Meldung produzieren.

Dies ist nur unter Verwendung von END= möglich. Also:
Regel: **Kein READ ohne END= !**

```
      PROGRAM Methode 1
*  Anzahl auf eine erste extra Zeile; dann die Daten beliebig
      INTEGER MaxAnzahl, Anzahl, i
      PARAMETER (MaxAnzahl = 100)
      INTEGER Feld(MaxAnzahl)
      READ(*,*,END=99) Anzahl
      IF( Anzahl .GT. MaxAnzahl ) THEN
         PRINT*,Anzahl,' Daten passen nicht in',MaxAnzahl,' Plätze'
         PRINT*,'Programm abgebrochen vor dem Einlesen der Daten'
         STOP
      ENDIF
      READ(*,*,END=99) (Feld(i),i=1,Anzahl)
      GOTO 20
   99 PRINT*,'Zu wenig Daten! Nur',i,' - 1 statt',Anzahl,': STOP'
      STOP
   20 PRINT*,i,' Daten gelesen einschl. Endekennung'
       continue
      END

      PROGRAM Methode 2
*  je 1 Datenelement pro Zeile, Einlesen bis Endekennung (hier 999999)
      INTEGER MaxAnzahl, Endekennung, i
      PARAMETER (MaxAnzahl = 100, Endekennung = 999999)
      INTEGER Feld(MaxAnzahl)
      DO i=1,MaxAnzahl
        READ(*,*,END=99) Feld(i)
        IF( Feld(i) .EQ. Endekennung ) GOTO 20
      ENDDO
      PRINT*,Maxanzahl,' Daten gelesen; Endekennung nicht gefunden'
      STOP
   99 PRINT*,'Zu wenig Daten: ',i,' - 1: Endekennung nicht gefunden'
      STOP
   20 PRINT*,i,' Daten gelesen einschl. Endekennung'
       continue
      END

      PROGRAM Methode 3
*  Daten beliebig, Einlesen bis EOF
      INTEGER MaxAnzahl,i
      PARAMETER (MaxAnzahl = 100)
      INTEGER Feld(MaxAnzahl)
      READ(*,*,END=99) (Feld(i)), i=1,MaxAnzahl)
      PRINT*,'Es wurden',MaxAnzahl,' Daten eingelesen'
      GOTO 20
   99 PRINT*,'Es wurden',i,' - 1 Daten eingelesen. Ende der Daten'
   20  continue
      END
```

Bemerkungen:
1. Die PARAMETER-Anweisung (s.a. Kap. 6 Beispielprogramm) ist bequem, um anschließend die Dimensionierung von Feldern zentral abändern zu können. Die Dimensionierung erfolgt in den obigen Beispielen mit dem "Parameter" (eigentlich: mit der Konstanten) MaxAnzahl.
2. Die CONTINUE-Anweisung ist eine sog. Leeranweisung (bewirkt nichts).
3. Die DO-Variable i, die die eingelesenen Daten zählt, hat beim Auftreten von EOF bereits um 1 weitergezählt und gibt in diesem Fall die Zahl der eingelesenen Daten + 1 an.
4. Warum haben wir nicht "PRINT*,'Es wurden ',i-1,' Daten gelesen' geschrieben, statt dessen oben immer '...',i,' - 1 Daten ...' ? In welchen Fällen führt ...,i-1,... zu einer Fehlermeldung?

Formatierte Eingabe: Die Grundgedanken

Die formatierte Eingabe (Standard-Eingabe Gerät Nr. 5 wie oben) hat die Form:

```
        READ(*,'( Formatinfo )') Variablen-Liste
```

oder

```
        READ(*,FMT=n) Variablen-Liste
    n   FORMAT( Formatinfo )
```

Die FORMAT-Anweisung ist wie eine Typ-Anweisung eine Definition (des Eingabevorgangs) und logisch eigentlich ein Teil der READ-Anweisung. Vergibt man eine Anweisungsnummer für eine extra FORMAT-Anweisung, so können mehrere READ-Anweisungen dieselbe FORMAT-Anweisung verwenden. Die FORMAT-Anweisung kann überall im Programm geschrieben werden. Empfehlenswert ist es, alle Formate am Anfang oder alle am Ende des Programms zu sammeln.

Das Format enthält im wesentlichen Information der Art:
"ab Spalte ... ist der Wert der nächsten Variablen aus den nächsten ... Spalten zu entnehmen"

Beispiel:
Einlesen einer ganzen Zeile (maximale Länge 80) in eine Variable vom Typ CHARACTER der Länge 80 (s. Kap. 8):

```
       CHARACTER*80 Zeile
       READ(*,100) Zeile
   100 FORMAT(A)
```

oder

```
READ(*, '(A)' ) Zeile
```

Der Buchstabe A als Formatinfo bedeutet für das READ:
lies eine neue Zeile von Anfang an als Character der Länge, wie es die Variable in der Liste (hier die Variable Zeile) angibt.
Es werden also in diesem Fall 80 Zeichen so in die Variable Zeile gestellt, wie sie am Bildschirm erscheinen.

Beim Vergleich formatierte Eingabe mit formatfreier Eingabe ergibt sich, daß das Schreiben größerer Datenmengen im freien Format Tippfehler verzeiht, die mit der nicht genau eingehaltenen Spalte für die einzelnen Daten entstehen. Bei numerischen formatierten Daten würden sich sonst katastrophale Veränderungen der betr. Werte ergeben. Nichtnumerische Daten (Textdaten) sind freilich in Hochkommata einzuschließen.
Aus diesem Grunde wird die Formatsteuerung im einzelnen erst im folgenden Kapitel beschrieben; die Logik der Ausgabe mit Format ist dann analog auf die Eingabe anzuwenden.

Übungen

1. Lesen Sie eine unbekannte Zahl von INTEGER-Werten (je 1 pro Zeile) und bestimmen Sie den größten und den kleinsten Wert. Drucken Sie jeden eingelesenen Wert und am Ende die beiden Ergebnisse.
   Version a) Verwenden Sie kein Feld
   Version b) Speichern Sie zuerst alle Werte in einem Feld.
   Achten Sie darauf, daß in keinem denkbaren Fall das Programm mit *ERR* abbricht, d.h. fangen Sie alle Fehlermöglichkeiten ab.

2. Schreiben Sie ein Programm zur Lösung des linearen Gleichungssystems

   $$Ax + By = C$$
   $$Dx + Ey = F$$

   Lesen Sie die Werte A,B,C aus der 1. Zeile, D,E,F aus der 2. Zeile.
   Testen Sie Ihr Programm mit einer unbekannten Anzahl von Zeilen.
   Die Lösung nach der Cramerschen Regel ist:
   $X = (CE - BF) / D$; $y = (AF - CD) / D$; $D = AE - BD$
   Vergessen Sie nicht den Fall Determinante D=0 (Nenner=0) !

Ausgabe

Die Antwort auf die 3 Fragen vom Anfang des Kapitels:
zu a):
Ausgabe-Geräte können sein:

Kartenstanzer, (Fernschreiber-) Lochstreifen (veraltet)
Bildschirm, Magnetband, Magnetplatte (typisch für größere Rechner)
Diskette, Datasette, Bildschirm / Fernseher (typisch für Microcomputer)

Im Kurs haben wir Bildschirm und Disketten / Festplatte(n) zur Verfügung. Mit entspr. Fortran-Anweisungen (OPEN) können wir diese Geräte mit dem Fortranprogramm verbinden.
Ohne besondere Anweisungen nimmt Fortran an, daß die Daten auf dem Bildschirm angezeigt werden: dies heißt "Standard-Ausgabe"; sie hat in Fortran als Geräte-Nummern 6 oder *, s.u.
Anstelle der allgemeinen WRITE-Anweisung mit Angabe der Geräte-Nr. kann "PRINT*" verwendet werden und bezieht sich dann nur auf die Standard-Ausgabe (Bildschirm als vorläufiges Medium statt Drucker).

zu b):
Die Ausgabe besteht aus einer Folge von Zeichen, die am Bildschirm sichtbar (auf dem Drucker druckbar) sind, sodaß diese auch direkt gelesen und verstanden werden können.
Im Idealfall müßten die eingelesenen Werte in derselben Form wieder ausgegeben werden bzw. die ausgegebenen Werte müßten wieder lesbar sein und den ursprünglichen Zustand der Werte der Variablen wieder herstellen.
Das ist tatsächlich bei der formatfreien Ein- und Ausgabe der Fall, wenn es sich um numerische (und logische) Daten handelt: Sie brauchen nur READ*, in PRINT*, oder umgekehrt zu ändern.
Bei Verwendung von Formaten ist das Vertauschen von READ und WRITE mit gewissen Einschränkungen verbunden (s.u. Stichwort Druckervorschubsteuerung), die ja logisch nicht umkehrbar ist.

Bei der Ausgabe vom Typ INTEGER oder Typ REAL müssen zusätzlich Dezimalpunkte, Exponenten, Vorzeichen usw. berücksichtigt werden. Wie das geschehen soll, ist von Fortran automatisch fest vorgegeben ("formatfreie Ausgabe") oder wird mit Hilfe eines Formats im Programm definiert ("formatierte Ausgabe"). Formatfreie Ausgabe kann ohne Änderung wieder im Freiformat eingelesen werden, s.o., außer Text, der bei der Eingabe in '...' eingerahmt werden muß.

zu c):
Dies wird ausschließlich durch das Programm bestimmt, nämlich durch die Liste in der WRITE-Anweisung bzw. der PRINT-Anweisung. Diese Liste kann nicht nur Variable (einschließlich Feldvariable und Feldelemente, eventuell innerhalb von impliziten DO-Schleifen) enthalten, sondern auch Konstante und Ausdrücke.

Formatfreie Ausgabe haben wir bisher schon mit der PRINT-Anweisung kennengelernt. Dabei spielte der genaue Ort (Spalte), wo die Daten geschrieben waren, keine Rolle, nur die Reihenfolge.
Bei der formatierten Ausgabe wird genau festgelegt, in welche Spalten welcher Zeile das Ausgabe-Element plaziert wird. Die Anweisung dazu ist die WRITE-Anweisung mit FORMAT-Anweisung (das Format kann auch Teil der WRITE-Anweisung sein, s.u.).

Formen der WRITE-Anweisung

Im folgenden geht die Ausgabe auf Bildschirm (Standard-Einheit 6):

formatfrei	**formatiert**
PRINT*, Variablen-Liste	WRITE(*,FMT=n) Variablen-Liste n FORMAT(Formatinfo)

dass. mit expliziter Geräte-Angabe * bzw. 5:

WRITE(*,*) Variablen-Liste	WRITE(UNIT=6,FMT=n) Variablen-Liste n FORMAT(Formatinfo)

oder kürzer

```
WRITE(UNIT=6,FMT='( Formatinfo )') Variablen-Liste
```

Es kann anstelle von FMT=n oben auch
FMT='(Formatinfo)'
geschrieben werden.
Die Variablen-Liste besteht aus einzelnen, durch Komma getrennten Konstanten, Variablen oder Ausdrücken.

Der erste * im formatfreien WRITE bedeutet "Standard-Ausgabe-Einheit", d.h. Einheit 6 (Bildschirm); der 2. Stern bedeutet "formatfrei".

Formatfreie Ausgabe

Die Daten werden durch Komma, Leerzeichen oder Zeilenende voneinander getrennt. Ein WRITE schreibt immer am Anfang einer neuen Zeile weiter; wenn eine Zeile nicht genügend Platz hat, werden weitere Zeilen geschrieben.
Formatfrei geschriebene Ausgabe kann formatfrei wieder eingelesen werden, indem man nur "PRINT" (bzw. WRITE") durch "READ" ersetzt und die Variablen-Liste beibehält (außer Strings, s.o.).

Formatierte Ausgabe

Das "Format", d.h. das hinter der Angabe FMT= stehende Formatinfo (bzw. das Formatinfo der FORMAT-Anweisung) beantwortet 3 Fragen:
1. Wie soll die Darstellung der Fortran-Variablen, -Konstanten sein? Soll eine INTEGER-Zahl mit oder ohne Vorzeichen ausgegeben werden? Wieviele Druckstellen (Plätze auf dem Bildschirm) soll sie belegen?
2. Ab welcher Spalte soll ausgegeben werden?
3. In welcher Zeile soll ausgegeben werden, wieviel Zeilen tiefer als die letzte Zeile ?

Ausgabe von INTEGER-Zahlen

Wir spielen die oben genannten 3 Aspekte am Beispiel der Ausgabe einer INTEGER-Zahl durch.

Der I-Code

Frage 1 wird mit der Angabe eines I-Codes (I wie INTEGER) beantwortet. Der Zahlenbereich geht von ca. +2 bis -2 Milliarden. Also kann eine INTEGER-Zahl bis zu 10 Ziffern und ein Vorzeichen, also max. 11 Plätze einnehmen. Oft weiß man aber, daß die Zahl kleiner ist und z.B. in 3 Plätzen darstellbar ist. Der I-Code enthält also noch die Anzahl der zu füllenden Plätze, z.B. "I3" für eine INTEGER-Zahl und für 3 Zeichen.

I-Code allg. Form: Iz belegt z Zeichen für eine INTEGER-Zahl

Der T-Code

Zur Klärung der 2. Frage teilen wir dem Compiler die Spalte mit, wo die Ausgabe beginnen soll. Beides zusammen:

```
   WRITE(*,FMT='( T2,I3 )' )  -85
```

Dies heißt: "Gib die Zahl -85 ab Sp. 2 mit insges. 3 Zeichen aus". In Sp. 2-4 kommt also -85 zu stehen.

Betrachten Sie die folgende Anweisung:

```
         WRITE(*, '( T3,I4,T8,I3 )' )  7294, -38

Ausgabeposition Spalte: 123456789012
Ausgabe:                  7294 -38
```

Der T -Formatcode zeigt auf die Position n, wo die Ausgabe beginnen soll; ab dieser Spalte belegt der I-Formatcode z Zeichen für die entspr. INTEGER-Zahl.

T-Code allg. Form: Tn zeigt auf Spalte n

Was passiert, wenn zuviele oder zu wenig Zeichen reserviert sind? Im ersten Fall wird links mit Leerzeichen erweitert; man sagt auch, die Zahl wird rechtsbündig in die z Positionen eingestellt. Im 2. Fall, wo die Zahl nicht vollständig darstellbar wäre, erscheint z mal ein *. Ein positives Vorzeichen wird nicht ausgegeben.

Beispiele:

```
                                  Ausgabe ( b steht für blank )
                                  Spalte 1: |
WRITE(6,FMT='( T4,I4 )') 12345        ****
WRITE(6,FMT='( T3,I5 )') 12345       12345
WRITE(6,FMT='( T2,I6 )') 12345      b12345
WRITE(6,FMT='(    I7 )') 12345     bb12345
WRITE(6,FMT='( T4,I4 )')-12345        ****
WRITE(6,FMT='( T3,I5 )')-12345       *****
WRITE(6,FMT='( T2,I6 )')-12345      -12345
WRITE(6,FMT='(    I7 )')-12345     b-12345
```

Übungen

1. Mit T75,I8: welche Spalte wird als letzte mit I8 belegt?
2. Schreiben Sie eine Folge von T- und I-Codes, um die Spalten 16-24, 25 und 80-90 für 3 INTEGER-Zahlen zu belegen.
3. Schreiben Sie eine FORMAT-Anweisung zu "WRITE(*,100) -123,4567,89", sodaß die Zeichen -123456789 in Spalte 8 bis 17 erscheinen.
4. Mit demselben WRITE sollen die Zeichen "4567 89 -123 " in Spalte 2 bis 15 erscheinen.
5. Was kommt heraus mit

```
       WRITE(6,100) 333,22,1
   100 FORMAT(T10,I3,T10,I2,T10,I1)
```

Der X -Code

Allg. Form: nX übergeht n Stellen

```
Z.B. hat  WRITE(6,'(   1X,   4X    ,I10)' )      dieselbe Wirkung wie
          WRITE(6,'( '' '',''    '',I10)' )           oder
          WRITE(6,'(      5X,      ,I10)' )           oder
          WRITE(6,'(      T6,      ,I10)' )
```

Die INTEGER-Zahl wird in Sp. 6-11 geschrieben, Sp. 1-5 werden blank.

Implizite Angabe der Ausgabe-Position

Ist kein T-Code vorhanden, so bestimmt sich die Ausgabe-Position implizit ab Sp. 1 (anfangs) bzw. im Anschluß an die vorhergehende Ausgabe. Die folgenden Formate haben alle die gleiche Wirkung:

```
     (T1,1X,T2,I6,T8,I5)          (T2,I6,T8,I5)
     (T1,1X,T2,I6,I5)             (5X,I6,T8,I5)
     (1X,I6,I5)
```

Es muß nicht unbedingt von links nach rechts geschrieben werden: z.B. die 1. INTEGER-Zahl in Sp. 8-12, die 2. in Sp. 2-7 erreicht man mit

```
     (T8,I5,T2,I6)
```

ASA-Drucksteuerzeichen

Die Schnelldrucker an IBM-Großrechnern (Banddrucker oder Laserdrucker im Zeilen-Druckmodus) interpretieren das 1. Zeichen in jeder Zeile als Steuerzeichen für den Papiertransport.
Diese sog. ASA-Steuerzeichen haben folgende Bedeutung:

1	"neue Seite"	am Anfang einer neuen Seite
+	"alte Zeile"	am Anfang der bereits bedruckten Zeile
Leerzeichen	"neue Zeile"	am Anfang der nächsten Zeile
0	"1 Leerzeile"	am Anfang der übernächsten Zeile
-	"2 Leerzeilen"	am Anfang der überübernächsten Zeile

Alle anderen Zeichen in Sp.1 werden als Leerzeichen betrachtet (ergibt also eine neue Zeile). Druckt man jedoch z.B. die Zahl 123 ab Sp. 1, so erscheint die Zahl 23 auf einer neuen Seite usw.
Man sorge also dafür, daß jede Zeile ein Leerzeichen in Sp. 1 besitzt, und daß die auszudruckenden Daten erst ab Sp. 2 beginnen. Dies erreicht man am einfachsten mit

```
       WRITE(*, FMT='( 1X, ...    )' ) ...
```

So wird bei jedem WRITE das Papier auf eine neue Zeile transportiert.
Eine neue Seite bzw. Doppeldruck erhält man mit

```
       WRITE(*,FMT='( ''1'', ...    )' )  bzw. FMT='( ''+'', ...
```

Alles in "FORMAT"-Schreibweise:

```
    WRITE(*,100)...  bzw.     WRITE(*,200)...  bzw.     WRITE(*,300)...
100 FORMAT( 1X,...)  bzw. 200 FORMAT('1',...)  bzw. 300 FORMAT('+',...)
```

Will man ASA-Steuerzeichen am PC verwenden, so ist eine OPEN-Anweisung mit einem Filenamen mit vorangestelltem "(C)" erforderlich, z.B.

```
     OPEN(6,FILE='(C)P1.OUT')    bzw.     OPEN(6,FILE='(C)TERMINAL')
```

für das File P1.OUT bzw. den Bildschirm.

Übungen

1. Schreiben Sie mindestens 2 weitere Formate mit der Wirkung von
   a) FORMAT('+',T5,I12)
   b) FORMAT(I14,T20,I3)
   c) FORMAT(I12,T1,'0',20X,I3)
2. Was kommt heraus mit (verifizieren Sie es am Computer):

```
   a)    WRITE(6,2) 1,2,3
       2 FORMAT(I2,2X,I4,2X,I3)
   b)    WRITE(6,5) 1,2,3
       5 FORMAT(I2,T14,I1,T6,I4)
   c)    WRITE(6,8) -1,-2
       8 FORMAT(I2,T1,I2)
   d)    N=17
         WRITE(6,14) N,MOD(18,N),MOD(N,18),N/3
      14 FORMAT(I2,2X,I4,2X,I3)
   e)    WRITE(6,16) 1234567,INT(SQRT(10000.))
      16 FORMAT(' ',I6,T4,I3)
```

Ausgabe von REAL-Zahlen

REAL-Zahlen können mit oder ohne Exponenten ausgegeben werden. Ohne Exponent ist der bei der Ausgabe darstellbare Zahlenbereich nur etwa +- 100000 ... 0.000001. Mit Exponent gibt es keine Einschränkungen.

Der F-Code

Der F-Code stellt die Zahlen ohne Exponent dar.

Allg. Form: Fz.d REAL-Zahlen, z Zeichen, d Dezimalen rechts vom .

Beispiele:

```
                                  Ausgabe ( b steht für blank )
                                  Spalte 1: |
WRITE(6,FMT='( F7.0 )') 123.45              bbb123.
WRITE(6,FMT='( F7.1 )') 123.45              bb123.4
WRITE(6,FMT='( F7.2 )') 123.45              b123.45
WRITE(6,FMT='( F7.3 )') 123.45              123.450
WRITE(6,FMT='( F7.4 )') 123.45              *******
WRITE(6,FMT='( F6.2 )') 123.45              123.45
WRITE(6,FMT='( F5.2 )') 123.45              *****
WRITE(6,FMT='( F4.2 )') 123.45              ****
```

Der E-Code

Der E-Code stellt Zahlen mit Exponent in einfacher Genauigkeit dar.

Allg. Form: Ez.d REAL-Zahlen, z Zeichen, d Dezimalen rechts vom .

Beispiele:

```
                                  Ausgabe ( b steht für blank )
                                  Spalte 1: |
WRITE(6,FMT='( E7.0 )') 123.45              *******
WRITE(6,FMT='( E7.1 )') 123.45              0.1E+03
WRITE(6,FMT='( E7.2 )') 123.45              .12E+03
WRITE(6,FMT='( E7.3 )') 123.45              *******
WRITE(6,FMT='( E7.4 )') 123.45              *******
WRITE(6,FMT='( E6.2 )') 123.45              ******
WRITE(6,FMT='( E9.4 )') 123.45              .1234E+03
WRITE(6,FMT='(E10.5 )') 123.45              .12345E+03
WRITE(6,FMT='(E11.5 )') 123.45              0.12345E+03
WRITE(6,FMT='(E12.5 )') 123.45              b0.12345E+03
WRITE(6,FMT='(E16.8 )') 123.45678           b0.12345680E+03
```

Man sieht, daß maximal 7 Dezimalen (mit Rundung) herauskommen. Die beiden Nullen im letzten Beispiel geben offensichtlich die Zahl 123.45678 (8 Dezimalen) nicht richtig wieder.

Der D-Code

Der D-Code stellt Zahlen mit Exponent in doppelter Genauigkeit dar.

Allg. Form: Dz.d REAL-Zahlen, z Zeichen, d Dezimalen rechts vom .

Beispiel:

```
                                  Ausgabe ( b steht für blank )
                                  Spalte 1: |
...,FMT='(D24.16 )')123.4567890123456         b0.1234568000000000E+03
...,FMT='(D24.16 )')123.4567890123456D0       b0.1234567890123460E+03
```

Im 1. Beispiel wird die REAL-Konstante trotz der vielen Ziffern nur als einfach genau betrachtet: erst der Zusatz "D0" macht eine doppelt genaue Konstante daraus mit maximal 15 Dezimalen.

Der G-Code

Der G-Code stellt die Zahlen entweder ohne Exponent oder automatisch, falls erforderlich, mit Exponent dar. Außerdem wird hier nicht die Zahl der Dezimalen rechts vom . angegeben, was ja, wie oben an den Beispielen ersichtlich, z.T. irreführende Ziffern zur Anzeige bringt.
Statt dessen wird die Zahl der signifikanten Dezimalen festgelegt, d.h. die Zahl aller angegebenen Dezimalen vor und hinter dem . ohne führende Nullen. Dies ist der für wissenschaftliche Berechnungen am besten geeignete Formatcode für REAL-Zahlen.

Allg. Form: Gz.s REAL-Zahlen, z Zeichen, s signifikante Dezimalen

Beispiele (identisch wie oben): Ausgabe (b steht für blank)
Spalte 1: │

```
wie oben mit F-Code:
WRITE(6,FMT='( G7.0 )') 123.45              *******
WRITE(6,FMT='( G7.1 )') 123.45              0.1E+03
WRITE(6,FMT='( G7.2 )') 123.45              .12E+03
WRITE(6,FMT='( G7.3 )') 123.45              ***
WRITE(6,FMT='( G7.4 )') 123.45              ****
WRITE(6,FMT='( G6.2 )') 123.45              ******
wie oben mit E-Code:
WRITE(6,FMT='( G9.4 )') 123.45              123.4bbbb
WRITE(6,FMT='(G10.5 )') 123.45              123.45bbbb
WRITE(6,FMT='(G11.5 )') 123.45              b123.45bbb
WRITE(6,FMT='(G12.5 )') 123.45              bb123.45bb
WRITE(6,FMT='(G16.8 )') 123.45678           bb123.456800000

wie oben (dort mit D-Code):
...,FMT='(G24.16 )')123.4567890123456789    bb123.4568000000000
...,FMT='(G24.16 )')123.4567890123456789D0  bb123.4567890123460
```

Übersicht über alle Formatcodes

Im folgenden gelten die Abkürzungen:

z: Anzahl der Zeichen insgesamt zur Darstellung der Zahl	(> 0)
d: Anzahl der Dezimalen rechts vom Dezimalpunkt	(>=0)
s: Anzahl der signifikanten Dezimalen (ohne führende Nullen)	(> 0)
w: Wiederholungsfaktor	(> 0)

Allgemein gilt: z,d,s müssen immer angegeben werden; w kann wegbleiben.

Für INTEGER-Zahlen:	Allg. Form	Darstellungsart
I-Code	wIz	ohne oder mit neg. Vorzeichen
	wIz.m	minimal m Ziffern (m <= z) d.h. evtl. mit führenden Nullen

Für REAL-Zahlen (komplexe Zahlen erfordern 2 Real-Formate):

F-Code	wFz.d	Gleitpunkt
E-Code	wEz.d	mit E-Exponent einfach genau
	wEz.dEe	dass., Exponent "E" mit e Stellen
D-Code	wDz.d	mit D-Exponent doppelt genau
	wDz.dDe	dass., Exponent "D" mit e Stellen

Für alle Zahlen unabhängig vom Typ und für logische Werte:

G-Code	wGz.s	Gleitpunkt oder mit Exponent
	wGz.sEe	dass., Exponent "E" mit e Stellen
	wGz.sDe	dass., Exponent "D" mit e Stellen

Dabei sollte z >= d+7 bzw. z >= s+7 sein, um Exponent mit Vorzeichen, Dezimalpunkt usw. unterzubringen; falls der Platz dafür und zusätzlich für den ganzzahligen Teil der Zahl nicht reicht, werden *** ausgegeben.

Der G-Code schaltet automatisch auf I (bei Typ INTEGER) oder auf F / E / D -Format um. Falls die Zahl im F-Format darstellbar ist, wird dieses verwendet, sonst das E bzw. D -Format je nach Typ. Außerdem ist hier s die Anzahl der signifikanten Dezimalen statt der Dezimalen rechts vom Dezimalpunkt; man kann z.B. ein nur auf 3 Dezimalen genaues Ergebnis auch mit 3 Dezimalen (ohne führende Nullen gerechnet) ausdrucken:
z.B. ergibt 0.0148 mit F8.3 -> 0.015, mit G8.3 -> .148E-01
aber 1.0148 mit F8.3 -> 1.015, mit G8.3 -> 1.01

Für Strings (Typ CHARACTER):

A-Code	wA	CHARACTER beliebiger Länge
	wAz	dass. mit Länge z linksbündig

Für logische Werte (Typ LOGICAL), s.a. G-Code:

L-Code	wLz	T für .TRUE., F für .FALSE.

Für hexadezimale Darstellung(auch in DATA-Anweisungen als Wert erlaubt):

Z-Code	Znn	2 Hexa-Ziffern (0-9,A-F) je Byte

Tabulator:

T-Code	Tn	positioniert auf Spalte n
	TLn	positioniert n Spalten nach links
	TRn	positioniert n Spalten nach rechts

Überspringen von Positionen:

X-Code	nX	geht n Positionen nach rechts

Skalenfaktor (D-,E-,F-,G-Code):

P-Code	iP	multipliziert bei der Ausgabe mit 10**i (i: Integer mit Vorzeichen)

Steuerung blank als 0 oder als nicht vorhanden (Voreinstellung: BZ)

BZ	blank im formatierten Eingabe-Feld wird als 0 interpretiert
BN	blank im formatierten Eingabe-Feld gilt als nicht vorhanden

Steuerung positives Vorzeichen (D-,E-,F-,G-,I-Code)

SP	erzeugt in jedem Fall ein +
SS	unterdrückt in jedem Fall ein +
S	überläßt es wieder dem Compiler

Steuerung der Ausführung von autonomen Codes

:	beendet autonome Codes, sobald die E/A-Liste zu Ende ist

Steuerung "neuer Record"

/	"neuer Record", meist gleichbedeutend mit "neue Zeile" Regel: gilt auch automatisch zu Beginn jeder E/A-Anweisung

Feinsteuerung der Bildschirmausgabe (Ausnahme von obiger Regel)

```
     $     Cursor bleibt in seiner Zeile stehen für die folgende Eingabe
           (Fortran90-Standard:   WRITE(...,ADVANCE='NO') ...
```

Beispiel:

```
   WRITE(*, '( 1X,A,I4 / 1X, A,I4 )' )  'i=',I,'j=',J
```

schreibt die Werte von I und J untereinander in 2 Zeilen.

```
   WRITE(*, '( 1X,A I4 // 1X,A,I4 )' )   'i=',I,'j=',J
```

erzeugt dazwischen eine extra Leerzeile.

Wiederholungsfaktoren, Code-Gruppen

ersparen das mehrmalige Hinschreiben desselben Formatcodes. Wiederholen kann man auch Gruppen in Klammern eingeschlossener Formatcodes, es ist

```
    5F6.1          dasselbe wie   F6.1,F6.1,F6.1,F6.1,F6.1     und
    3(F6.1,I3)     dasselbe wie   F6.1,I3,F6.1,I3,F6.1,I3
```

Insgesamt sind 7 Klammerebenen erlaubt innerhalb der äußeren Klammer.

Beispiel: Ausdruck eines Feldes von 100 Elementen in der Form
F(0)=wert F(1)=wert ... F(99)=wert, je 4 Werte pro Zeile

```
      WRITE(6,100) ( '  F(',I, ')=', F(I), I=0,99 )
100   FORMAT(    4 (  A    ,I2, A  , G12.5 : ) )
```

Der : verhindert, daß am Ende eine Zeile mit "F(" erzeugt wird.

Übungen

1. Was ergibt sich mit (nachprüfen am Computer!):
   a) WRITE(*,'(I7)') -12345
   b) WRITE(*,'(I5)') -12345
   c) WRITE(*,'(F8.2)') 12345.
   d) WRITE(*,'(F7.1)') -123.45
   e) WRITE(*,'(F9.4)') .00024E-02
   f) WRITE(*,'(E10.2)') -.00025
   g) WRITE(*,'(E12.5)') 2.0006
   h) WRITE(*,'(F8.0)') 10.E5
   i) WRITE(*,'(F9.7)') 10.E-10
   j) WRITE(*,'(I4)') 2.64
   k) WRITE(*,'(F4.0)') 3
2. Ersetzen Sie in 1. c) bis i) und k) jeweils im Formatcode die Buchstaben E und F durch G. Was ändert sich?
3. Bestimmen Sie die Startposition für jedes der folgenden Formatcodes:
   a) FORMAT('0',T5,I8,T2,I3)
   b) FORMAT(F12.7,3X,T20,F8.0,T17,'X=')
   c) FORMAT('1X(I)='/E15.6,T1,'12345')
   d) FORMAT(T5,I8,2X,E15.6,T1,'12345')
4. Gegeben sei FORMAT('0',2(3I2,3X),3F6.2,3(2X,F8.0)).
   a) Wieviele Gruppen enthält das Format?
   b) Wieviele Formatcodes enthält das Format?
   c) Welches ist der 5. Formatcode?
   d) Welche Spalten gehören zum ersten F8.0?
   e) Welche Spalten gehören zum 13. und 14. Formatcode?
   f) Welche Spalten gehören zum dritten F6.2?
   g) Wenn das 5. Datenelement REAL wäre, würde sich ein Error ergeben?
   h) Mit dem 8. Datenelement 12345.67, welche Zeichen werden erzeugt?
5. Was kommt heraus mit X=+49 und I=-25 und
   a) WRITE(*,'(''1'',F3.0,T1,I4)') X**(1./2.),I
   b) WRITE(*,'(''1'',2(3X,2F4.1,I4,2I5)') 2.,3.,4,5,6,7.,8.,9,10,11

Aufgaben

1. Verwenden Sie formatierte Ausgabe für Kap. 5 Aufg. 5 sowie REAL-Zahlen für die Temperaturen usw.. Die Ergebnisse sollen so aussehen:

	Mo	Di	Mi	Do	Fr	Sa	So	Durch.	Max	Min
Woche 1	x	x	x	x	x	x	x	x	x	x
Woche 2	x	x	x	x	x	x	x	x	x	x
Woche 3	x	x	x	x	x	x	x	x	x	x
Woche 4	x	x	x	x	x	x	x	x	x	x
4-Wochen-Statistik								x	x	x

   Die 4 Zeilen "Woche 1 ..." sollen mit DO i=1,4 und einer impliziten DO-Schleife geschrieben werden.
2. Erweitern Sie die formatierte Ausgabe für eine Tabelle der Werte für Kap. 5 Aufg. 6 (Tagesstatistik).
3. Verwenden Sie die Subroutine von Kap. 6 Aufg. 9 für die folgende Aufgabe: Mergen von k 5 Mengen. (man nehme k <= 5 an).
   Das Hauptprogramm soll zunächst den Wert von k einlesen, sodann die ersten beiden Mengen, diese mergen, dann die nächste Menge, diese mergen usw. Die 1. Datenzeile soll jeweils die Zahl der folgenden Eingabe-Datenelemente enthalten (die allererste enthält k).

Zusammenfassung: Allgemeine Regeln für Formate

Man unterscheidet autonome Formatcodes und beschreibende Formatcodes:
Autonome Formatcodes sind solche, die allein eine Wirkung haben:
/ "neuer Record", T "Tabulator", X "Leerstelle", '..' "String-Konstante"
: "Ende Autonomie, wenn E/A-Liste erschöpft", $ "Cursor bleibt stehen"
BN BZ "blank = nichts" "blank = 0", SP SS "+ ausgeben" "+ unterdrücken"
Alle anderen Formatcodes sind beschreibende Formatcodes: Diese sind nur in Verbindung mit der E/A-Liste wirksam, also in Verbindung mit einer Variablen, Konstanten oder einem Ausdruck (für eine WRITE-Anweisung bzw. PRINT-Anweisung) und in Verbindung mit einer Variablen (READ-Anweisung).

Der Mechanismus läuft dann folgendermaßen ab:

1. Zuerst wird ein E/A-Befehl ausgelöst, d.h. "neuer Record" (i.a. praktisch gleichbedeutend mit "neuer Zeile")

2. Dann werden die Formatcodes der Reihe nach betrachtet:
   a) handelt es sich um einen autonomen Formatcode, wird er ausgeführt.
   b) wird ein beschreibender Formatcode gefunden, wird aus der E/A-Liste der Reihe nach ein Element (eine Variable) zugeordnet.

3. Zu der Kombination "beschreibender Formatcode" und "Variable / Konstante / Ausdruck" werden gemäß der Größe z (aus dem Formatcode) z Zeichen auf dem Bildschirm / aus der Eingabezeile bzw. z Zeichen auf dem Papier / in die Ausgabezeile bestimmt.

4. Dies wiederholt sich solange, bis die E/A-Liste abgearbeitet ist.

5. a) Die anschließend auftretenden autonomen Formatcodes werden zusätzlich ausgeführt bis zum nächsten beschreibenden Code.
   b) Ab dem Formatcode : wird dieser Vorgang unterbunden.

6. Enthält die E/A-Liste weniger Variable als beschreibende Formatcodes (mit Wiederholungsfaktor bzw. Gruppenwiederholungsfaktor gerechnet), so werden die überzähligen beschreibenden Formatcodes ignoriert.

7. Enthält die E/A-Liste mehr Elemente als beschreibende Formatcodes, dann wird weiter fortgefahren:
   a) falls keine zusätzliche Klammerung vorhanden ist, wieder von vorn (d.h. mindestens mit neuem E/A-Befehl "neuer Record")
   b) falls zusätzliche Klammerung vorhanden ist, bei derjenigen Klammer-auf, die zu der letzten inneren (zusätzlichen) Klammer-zu gehört, einschließlich Gruppenwiederholungsfaktor, falls vorh.

Zusätzliche E/A-Befehle ("neue Zeile") können als "/" eingefügt werden, z.B. wenn mit einem Format mehr als 80 Zeichen bearbeitet werden sollen (der Normal-Record (Zeile) hat für Ein- und Ausgabe die Länge 80).

Tabellarische Übersicht

Beschreibung der unten verwendeten Begriffe:

Eingabe-Datenelement vom Formatcode erfaßte Eingabe-Zeichen. Sie werden zu einer einzigen Größe (Zahl, String) gemacht.
Eingabe-Variable Variable aus der Eingabe-Liste
erscheint intern ... computerspezifische Darstellung der Typen INTEGER, REAL, COMPLEX, LOGICAL
ist zulässig intern wie oben
Ausgabe-Element Variable, Konstante oder Ausdruck der Ausgabe-Liste
erscheint extern ... Darstellung auf dem externen Medium wie z.B. Bildschirm, Drucker, Magnetplatte usw.

Die beschreibenden Codes (Ausgabe):

mit For-mat-Code	ist zulässig als Eingabe-Datenelement	falls entspr. Eingabe-Variable	erscheint intern als	Sonderfälle: wenn Zeichenzahl Formatcode > Länge der CHARACTER-Variable	Sonderfälle: wenn Zeichenzahl Formatcode < Länge der CHARACTER-Variable
A	beliebig (character)	beliebig	wie extern (character)	automatische Anpassung an Länge der Eingabe-Variablen	
An	beliebig (character)	beliebig	wie extern (character)	rechts wird abgeschnitten	rechts mit blank gefüllt
D	real*8	real*8	real*8	für alle numerischen Formate:	
E,F	real*4	real*4	real*4		
G	real integer	real integer	real integer	der zulässige Bereich für die entspr. Konstanten ist zu beachten	blank wird als 0 interetiert, falls nicht Formatcode BN wirksam
I	integer	integer	integer		
G,L	logical	logical	logical	erstes T oder F liefert Wert	

Die beschreibenden Codes (Eingabe):

mit For-mat-Code	ist zulässig intern für Ausgabe	falls entspr. Ausgabe-Element	erscheint extern als	Sonderfälle: wenn Zeichenzahl Formatcode > Zeichenzahl Ausgabe-Element	Sonderfälle: wenn Zeichenzahl Formatcode < Zeichenzahl Ausgabe-Element
A	beliebig (character)	beliebig	wie intern (character)	automatische Anpassung an Länge des Ausgabe-Elements	
An	beliebig (character)	beliebig	wie intern (character)	rechts mit blank gefüllt	rechts wird abgeschnitten
D	real*8	real*8	real mit D	links wird mit blank gefüllt	rechts wird abgeschnitten und gerundet
E	real*4	real*4	real mit E		
F	real*4	real*4	Gleitpunkt		
G	real	real	Gleitpunkt bzw. real mit D/E		falls kein Platz für INTEGER-Teil: Ausgabe von Sternchen *******
	integer	integer	integer		
I	integer	integer	integer		
G,L	logical	logical	T oder F	---	---

Kapitel 8: Der Typ CHARACTER

Fragen

1. Was versteht man unter dem Typ CHARACTER, was unter einem String?
2. Wie werden Strings definiert und verarbeitet?
3. Welche Operatoren und Funktionen gibt es zur Stringverarbeitung?

CHARACTER-Werte

Einzelne Zeichen (engl. character) oder Zeichenketten (engl. strings), z.B. "1234", können als Zahl interpretiert werden (mit arithmetischen Operationen) oder als Folge von 4 einzelnen Zeichen. Im ersteren Fall kann "1234" in einem INTEGER- oder REAL- Ausdruck ohne weiteres auftauchen; sonst muß die Zeichenkette als CHARACTER deklariert werden. Als Werte sind alle Zeichen erlaubt, die durch die Tastatur dargestellt werden können; genauer: es sind alle 256 Bitkombinationen möglich, die ein Byte annehmen kann.

Konstante, Variable, Ausdrücke

Eine CHARACTER-Konstante wird geschrieben als eine in Hochkomma eingeschlossene Zeichenkette, also z.B. 'a', 'Max', '15.3.1989'.
Bis zu 255 Zeichen sind zwischen den '...' zugelassen. Falls die Zeichenkette selbst das Hochkomma enthalten soll, muß dieses doppelt hingeschrieben werden, also z.B. 'tu''s nicht!', was eine Länge von 11 Zeichen ergibt. Ausprobieren: PRINT*,'tu''s nicht'

CHARACTER-Variable werden mit dem Typ CHARACTER deklariert, z.B.

```
CHARACTER A,B*2
CHARACTER *4 T, U*2, L*6/'LAENGE'/
CHARACTER MONAT(4)*4 /'Jan.','Feb.','Mär.','Apr.'/
```

Als Länge kann "*1" bis "*255" verwendet werden; "*1" kann wegbleiben.

Zuweisung:

```
CHARACTER A*n, B*m          n,m sind Zahlen von 1 bis 255
A = B              oder A = CHARACTER-Ausdruck
```

Ist bei Zuweisung B kürzer als A, wird rechts mit Leerzeichen auf die Länge von A aufgefüllt; ist A kürzer, wird rechts abgeschnitten.
Also: mit A*2 und B*4 wird durch A = 'Otto', B = 'ok' der Inhalt von
A: 'Ot', B: 'ok '

CHARACTER-Ausdrücke können mit den folgenden CHARACTER-Operatoren gebildet werden:

1. Substring-Angabe (wird hinter den entspr. String gestellt):
   String (*von* : *bis*)
   von, *bis* sind INTEGER. Sie geben den Ausschnitt an, wobei ab der 1. Position die einzelnen Zeichen gezählt werden. Von bzw. bis können wegbleiben mit der Bedeutung "von vorn" bzw. "bis hinten":
   String(:*bis*) bzw. String(*von*:)

2. Konkatenierungs-Operator // (zwei Schrägstriche)
   String1 // *String2* bzw. ... // ... // ... mit mehreren Strings bilden einen einzigen zusammengesetzten String.

Beispiele:
Es sei CHARACTER A, Name*8 /'Otto'/ deklariert. Dann ist
Name dasselbe wie Name(:) oder Name(1:) oder Name(:4) oder Name(1:4)

```
Name(1:2) = Name(3:4)   ->  'toto'       als Inhalt von Name
'L' // Name             ->  'LOtto'
Name = Name(:1)//' '//Name(2:2)//' '//Name(3:3)//' '//Name(4:)
```

Eingebaute Funktionen zur Stringverarbeitung

Fortran stellt folgende Funktionen bereit, z.T. vom Typ CHARACTER;
im folgenden bedeutet "String" eine CHARACTER -Konstante, -Variable,
-Feldelement oder ein hieraus gebildeter Substring, z.B. Name(3:4):

1. INTEGER FUNCTION INDEX (*String1*, *String2*)
   Funktionswert ist der INTEGER-Wert n, der das n.te Byte zählt,
   wo String2 (von links her gesehen) beginnt.
   Ist String2 nicht in String1 enthalten, wird 0 zurückgegeben.

2. INTEGER FUNCTION LEN (*String*)
   Funktionswert ist die Länge von String.
   Dies ist wichtig innerhalb eines Unterprogramms, um die Länge eines
   Parameters vom Typ CHARACTER zu bestimmen, sodaß dann Substring
   oder Konkatenierung angewandt werden können.

3. INTEGER FUNCTION ICHAR (*zeichen*)
   Funktionswert ist die zum Zeichen gehörige Byte-Code-Zahl 0...255
      'a' => 97, 'A' => 65, '0' => 48 im ASCII-Code (PC),
      'a' => 129, 'A' => 193, '0' => 240 im EBCDI-Code (IBM-Großrechner)

4. CHARACTER FUNCTION CHAR (*integerzahl*)
   Funktionswert ist die Umkehrung von ICHAR:
   zurückgegeben wird das Zeichen, das zur entspr. Code-Zahl gehört.
   Möglich für integerzahl sind die Zahlen 0 bis 255.

Wie man sieht, sind die Ergebnisse des Vergleichs zweier Strings
ganz verschieden, je nachdem man am PC oder Workstation (ASCII-Code)
oder am IBM-Großrechner (EBCDI-Code) sitzt; z.B. Sortieren wäre so in
Fortran vom verwendeten Computer abhängig.
Um dem abzuhelfen, gibt es noch die folgenden logischen Funktionen:

   LGE(*String1*,*String2*) liefert .TRUE. falls *String1* >= *String2*
   LGT(*String1*,*String2*) liefert .TRUE. falls *String1* > *String2*
   LLE(*String1*,*String2*) liefert .TRUE. falls *String1* >= *String2*
   LLT(*String1*,*String2*) liefert .TRUE. falls *String1* < *String2*

Maßgeblich für diese Funktionen ist der ASCII-Code.

Wenn man also z.B. statt IF (NAME1 .GE. NAME2) ...
schreibt: IF (LGE(NAME1,NAME2)) ...
dann wird die Stringverarbeitung auf allen Rechnern gleich laufen.

Beispiel 8.1

Wir lesen eine Zeile von 80 beliebigen Zeichen ein. Sie kann einzelne
oder mehrere Leerzeichen hintereinander enthalten; dadurch sollen ein-
zelne Worte getrennt werden. Wir wollen 1 Wort pro Zeile ausdrucken;
außerdem sollen mehrere hintereinander stehende Leerzeichen durch ein
einziges ersetzt und die Zeile in dieser Form ausgegeben werden.

```
      PROGRAM P81
* Beispiel 8.1: Worte trennen
* liest eine Zeile Länge 80 ein, gibt die einzelnen Worte getrennt aus
* und gibt außerdem die Zeile mit nur je 1 trennenden Leerzeichen aus
      IMPLICIT NONE
      CHARACTER *80 Zeile, Wort, hole_Wort_aus, Liste(40), NeueZeile
      INTEGER Zeiger, IndexU, Len_Trim, i,j,k
* Eingabe:
      PRINT*,'Bitte Zeile eintippen'
      READ (UNIT=5,FMT='(A)') Zeile
      PRINT*,'>>> Eingabezeile war:'
      PRINT*,Zeile
```

```
* Verarbeitung:
      Zeiger = IndexU( Zeile,' ' )
      IF (Zeiger .EQ. 0) STOP
      DO I=1,80
         Wort = hole_Wort_aus( Zeile(Zeiger:) )
         Liste(I) =  Wort
         Zeiger = Zeiger + Len_Trim(Wort)
         IF ( Wort .EQ. ' ' .OR. Zeiger .GT. 80 ) QUIT
         Zeiger = Zeiger + IndexU( Zeile(Zeiger+1:), ' ')
      ENDDO
* Ausgabe:
      PRINT*,'>>> Worte untereinander:'
      WRITE(6,'(A)') (Liste(K),K=1,I-1)
      NeueZeile = Liste(1)
      DO J=2,I-1
         NeueZeile = NeueZeile(1:Len_Trim(NeueZeile)) // ' ' // Liste(J)
      ENDDO
      PRINT*,'>>> Zeile mit nur 1 Leerzeichen zwischen den Worten:'
      WRITE(6,'(A)') NeueZeile
      END

      CHARACTER FUNCTION hole_Wort_aus *80( Puffer )
* ein Wort max. Länge 80 aus Puffer ohne leading und trailing blanks
      IMPLICIT NONE
      CHARACTER Puffer *(*)
      INTEGER Anfang, Ende, IndexU
      hole_Wort_aus = ' '
      Anfang = IndexU( Puffer,' ' )
      IF ( Anfang .EQ. 0 ) RETURN
      Ende   = INDEX ( Puffer(Anfang:),' ')
      IF ( Ende   .EQ. 0 ) Ende = LEN(Puffer)
      hole_Wort_aus = Puffer( Anfang:Ende )
      END

      INTEGER FUNCTION IndexU( String1,String2 )
* Wie eingebaute Funktion INDEX, aber mit prüfen auf ungleich
      CHARACTER *(*) String1, String2
      INTEGER L1, L2, I
      L1 = LEN(String1)
      L2 = LEN(String2)
      IF ( L1 .GE. L2 ) THEN
        DO I=1,L1-L2+1
          IF( String1( I : I+L2-1 ) .NE. String2) THEN
            IndexU = I
            RETURN
          ENDIF
        ENDDO
      ENDIF
      IndexU = 0
      END
      INTEGER FUNCTION Len_Trim( C )
* Länge eines CHARACTER-Strings ohne trailing blanks, minimal 1
      IMPLICIT NONE
      CHARACTER C *(*)
      INTEGER I
      DO I=LEN(C),1,-1
        IF ( C(I:I) .NE. ' ') THEN
          Len_Trim = I
          RETURN
        ENDIF
      ENDDO
      Len_Trim = 1
      END
```

Kommentar:

1. Zunächst fällt eins ins Auge: Die Aufgabe läßt sich eindeutig in 5 Zeilen beschreiben, zur Programmierung ist jedoch ein größerer Aufwand erforderlich. Dies ist typisch für Aufgaben mit Strings im weiteren Sinne (Textverarbeitung) und gilt nicht nur für Fortran.
2. Schon zur Lösung solcher doch einfacher Aufgaben ist es zweckmäßig, die Funktionen zur Stringverarbeitung durch selbst geschriebene zu ergänzen (hole_Wort_aus, IndexU, Len_Trim).
3. Das Programm ist klar gegliedert mit Kommentaren wie
   * Eingabe: --- ... * Verarbeitung: --- ... * Ausgabe: ---- ...
4. Besonders sei das Studium der Führung des Zeigers empfohlen. Dazu die folgenden Erläuterungen:

   Der Zeiger soll für den Aufruf von hole_Wort_aus jeweils auf dem Anfang eines Wortes stehen; zunächst also auf dem 1. Wort:

```
Zeiger = IndexU( Zeile,' ' )
```

   Die nächste Zeile dient dazu, den Sonderfall zu behandeln, daß die ganze eingegebene Zeile aus Leerzeichen besteht.

   Nachdem in der DO-Schleife ein Wort geholt wurde, wird mit

```
      Zeiger = Zeiger + Len_Trim(Wort)
```

   auf das 1. Leerzeichen hinter dem gefundenen Wort gesetzt.

   Die nächste Zeile beendet die DO-Schleife, falls der Rest der Zeile nur aus Leerzeichen besteht oder falls der Zeiger auf 81 steht, was dann passieren kann, wenn das letzte Wort bis einschl. Zeichen 80 geht. QUIT ist ein Vorgriff auf den Fortran90-Standard und macht hinter der Anweisung "ENDDO" weiter, dh. verläßt die DO-Schleife (im Fortran90 heißt die entspr. Anweisung EXIT).
   Ohne QUIT hätte man schreiben müssen:

```
            IF ( Wort .EQ. ' ' .OR. Zeiger .GT. 80 ) GOTO 77
            Zeiger = Zeiger + IndexU( Zeile(Zeiger+1:), ' ')
         ENDDO
77       CONTINUE
```

   Anschließend soll der Beginn des nächsten Wortes gefunden werden.

```
      Zeiger = Zeiger + IndexU( Zeile(Zeiger+1:), ' ')
```

   IndexU sucht ab dem Zeichen hinter dem, auf welches Zeiger zeigt

```
      ...(Zeile(Zeiger+1:),...
```

   nach einem Zeichen ungleich Leerzeichen; durch Addition wird der Zeiger auf dieses Zeichen gesetzt. Wenn kein solches Zeichen gefunden wird, liefert IndexU 0, d.h. der Zeiger bleibt ungeändert. Das nächste Wort, das hole_Wort_aus holt, wäre dann leer - siehe Anweisung "hole_Wort_aus = ' '" in der FUNCTION hole_Wort_aus.

Aufgaben

1. Erweitern Sie obiges Programm, sodaß als Trennzeichen für Worte nicht nur Leerzeichen, sondern alle Sonderzeichen betrachtet werden.
2. Lesen Sie wieder eine Zeile mit bis zu 80 Zeichen ein. Erstellen Sie eine Häufigkeitsstatistik der Buchstaben, Ziffern und Sonderzeichen.
3. Erweitern Sie die Aufgaben 1 und 2, daß nicht nur 1 Zeile, sondern beliebig viele bearbeitet werden; benutzen Sie zur Erkennung des Endes der Eingabe (der letzten zu verarbeitenden Zeile) eines der Verfahren von Kap. 6 (Einlesen einer unbekannten Zahl von Daten).
4. Erweitern Sie Aufgabe 3 zur Erstellung eines sog. KWIC-Index (KeyWordInContext): Zeilen der Länge 80 werden eingelesen; die einzelnen Worte sollen in Sp. 31 untereinander ausgegeben werden, zusätzlich mit maximal je 30 Zeichen links und rechts vom Wort.
5. Geben Sie den KWIC-Index nach den Worten (ab Sp. 31) sortiert aus.
6. Wie 4. und 5., aber das Zeilenende soll den Kontext nicht begrenzen (Volltext-KWIC).

Interne Files

Wir schreiben an die Stelle der Geräte-Nr. bei den E/A-Anweisungen READ und WRITE eine Variable vom Typ CHARACTER, auch "internes File" genannt. Damit kann man einen Typ in einen anderen konvertieren, wobei alle Methoden der Formatierung anwendbar sind.

Beispiel 8.2

```
      PROGRAM P82
* Beispiel 8.2: READ mit Internem File
* Wir wollen die Eingabe analysieren und je nach Bedarf
* anschließend als Integer oder als Real formatieren.
      CHARACTER Eingabe*11
      PRINT*,'bitte "I" und eine INTEGER-Zahl'
      PRINT*,' oder "R" und eine REAL-Zahl eingeben'
      READ(6,'(A)') Eingabe
      IF (Eingabe(1:1) .EQ. 'I' ) THEN
         READ ( UNIT=Eingabe(2:11), FMT='(I10)') IVariable
         PRINT*,'die INTEGER-Zahl war',IVariable
      ELSEIF (Eingabe(1:1) .EQ. 'R' ) THEN
         READ ( UNIT=Eingabe(2:11), FMT='(F10.0)') RVariable
         PRINT*,'die REAL-Zahl war',RVariable
      ELSE
         PRINT*,'bitte nur "I" oder "R"'
      ENDIF
      END
```

Kommentar:
Die Eingabezeile wird zunächst als ganzes (maximal 11 Zeichen) im A-Format in die CHARACTER-Variable INPUT eingelesen. A-Format heißt: keine Konvertierung der Eingabe.
Je nach "I" oder "R" wird dann mit dem entspr. Format die richtige Konvertierung vorgenommen, d.h. aus der eingetippten Zeichenfolge wird eine INTEGER- bzw. REAL-Zahl in interner Darstellung gemacht.
Der umgekehrte Vorgang ist ebenfalls möglich; er bietet sich an, um das Problem zu lösen, daß ein Format keine Variablen enthalten kann; z.B. wäre es angenehm, je nach Bedarf eine variable Zeichenbreite z auszurechnen und das Format entsprechend arbeiten zu lassen; dafür

Beispiel 8.3

```
* Beispiel 8.3: WRITE mit Internem File
* Erstellen eines "variablen" Formats
      CHARACTER VFORM*5 /'(Ixx)'/
* hier könnte die Berechnung der Größe z für das I-Format folgen ...
* wir setzen hier IZ einfach auf 8
      IZ = 8
      WRITE ( VFORM(3:4), '(I2)') IZ
* jetzt steht ' 8' anstelle von xx in VFORM!
      WRITE(6, VFORM ) ...
*     WRITE(6,'(I 8)') ... wird jetzt ausgeführt!
```

Die WRITE-Anweisung mit einer CHARACTER-Variablen als "Ausgabe-Gerät" konvertiert die INTEGER- (Binär-) Zahl IZ mit dem Formates I2 in den Typ Character, d.h. in unserem Fall in ein Leerzeichen und die 8. Diese 2 Zeichen werden in die "Unit" VFORM(3:4) geschrieben.
Neu ist in diesem Zusammenhang auch, daß anstelle des Formats auch eine CHARACTER-Variable angegeben werden kann (bisher haben wir ja nur die CHARACTER-Konstante '(...)' als Format benutzt).
Eine CHARACTER-Variable als "Eingabe-Gerät" oder als "Ausgabe-Gerät" stellt einen Record (eine Zeile) dar.

Ein File aus mehreren Records (Zeilen) wird mit einem CHARACTER-Feld geschrieben, z.B. kann CHARACTER*80 IFILE(100) ein internes File mit 100 Zeilen der Länge 80 darstellen. Das Schreiben und Lesen in und aus einem internen File spricht die einzelnen Records (Feldelemente) wie üblich mit dem / -Formatcode an (bzw. "neuer Record" bei Wiederholung des Formats, s. Kap. 7 oben).

Beispiel 8.4 (WRITE)

Wir nehmen das Beispiel von Kap. 7 zum Schreiben eines Feldes von 100 Elementen; wir wollen nachträglich einige Werte mit * kennzeichnen, die nach einer zusätzlichen Analyse als bedenklich gelten sollen.
Wir gehen so vor, daß wir das Feld wie oben schreiben, nur nicht auf die Geräte-Nr. 6, sondern in ein internes File. Anschließend bringen wir die nötigen Zusätze an den entspr. Stellen an (mit einfachen Zuweisungen) und schreiben dann das Ganze auf die Geräte-Nr. 6. Also:

```
* Beispiel 8.4: WRITE mit Internem File
* Erstellen einer Seite Ausgabe zunächst im Speicher
* mit der Möglichkeit, bel. Korrekturen anzubringen
      CHARACTER IFILE*80(25)
      WRITE(IFILE,100) ( 'F(',I, ')=', F(I), I=0,99 )
*statt WRITE(6,100) ( '  F(',I, ')=', F(I), I=0,99 ) oben
100   FORMAT(    4 (  A  ,I2, A  , G12.5 : ) )
* hier könnte die oben erwähnte Analyse folgen
* das Ergebnis soll sein, daß die Werte F(5) und F(6) markiert werden
* diese Werte stehen ja in der 2. Zeile Spalte 1-20 und 21-40, also:
       IFILE(2)( 1: 2)='**'
       IFILE(2)(21:22)='**'
* es wird alles, da schon formatiert (lesbar), im A-Format ausgegeben
       WRITE(6,'(A)') IFILE
* so einfach ist das?!
```

Auf diese Weise ist es in Fortran möglich, sowohl probeweise die Eingabe erst einmal im A-Format einzulesen, zu analysieren und mit dem passenden Format weiterzuverarbeiten (in eine entspr. interne Form bringen), wie im Beispiel 8.2, als auch die Ausgabe erst einmal in ein internes File zu schreiben und dann beliebige Korrekturen durch direkten Zugriff auf jedes einzelne Zeichen vorzunehmen (Zuweisung an einen Substring, z.B. INFILE(10)(20:30)='...' verändert "Zeile" 10 Spalte 20 bis 30) und am Ende das ganze interne File im A-Format auszugeben, wie in den Beispielen 8.3 und 8.4.

Ein letztes Beispiel

Beispiel 8.5

Aufgabenstellung:
Man schreibe eine Subroutine, die eine beliebige Funktion auf dem Bildschirm "plottet".

Dazu folgende Überlegungen:
Am PC hat man im Textmodus 25 Zeilen a 80 Zeichen zur Verfügung.
Mit passenden IBM-Grafikzeichen kann man jedoch an jeder Stelle unten, mitte, oben unterscheiden, hat dann also 75 mal 80 "Punkte".
Es sollen 3 Zeichensätze (Punkt, Strich, Quadrat) verwendet werden.
Für die 3 Zeichensätze bieten sich die folgenden Zeichen an:
1) CHAR 46,249,96; 2) CHAR 95,196,126; 3) CHAR 220,254,223,
wo der 1. Wert unten, der 2. mitte, der 3. oben darstellt.
Es sollen 76 Funktionswerte von 0 bis 7.5 in Schritten von 0.1 dargestellt werden mit entspr. beschrifteten Achsen.
Als Option sollen mehrere Kurven (mit neuem Zeichensatz) übereinander dargestellt werden.

Achtung: An jeder Stelle des Textbildschirms kann selbstverständlich jeweils nur 1 Zeichen stehen, da wir keine Zeichen zur Verfügung haben, die je 2 oder alle 3 Versionen eines Punktes gleichzeitig darstellen können. Ein Punkt einer Kurve wird auf jeden Fall durch einen neuen gelöscht, wenn der neue Punkt an derselben Stelle des Textschirms steht wie der alte (z.B. wenn der alte Punkt "unten" war und der neue "oben").

Man teste mit der Funktion 2*sin(x).

```
      PROGRAM P85
* Beispiel 8.5: Plot beliebiger Funktionen im PC-Textmodus
*               mit 3-facher Auflösung (80 x 75 "Punkte")
      IMPLICIT NONE
      INTEGER Ausgabe, Zeichensatz1, Zeichensatz2, Zeichensatz3
      LOGICAL Neu,Alt
      PARAMETER (Ausgabe = 6, Neu = .TRUE., Alt = .FALSE.,
     &    Zeichensatz1 = 1, Zeichensatz2 = 2, Zeichensatz3 = 3)
      EXTERNAL SIN2
*     external   falls eigene Funktion gemeint ist
*     intrinsic  für eine Fortran-Funktion (z.B. sin )

*     Aufruf des Plot-Programms
      CALL PLOT ('2*sin(x)', SIN2, Zeichensatz3, Neu)
*     nach dem Betrachten des Plots muß Enter gedrückt werden
      READ*
      WRITE ( UNIT=Ausgabe, FMT=*) 'Ende des Plots'
      END

      REAL FUNCTION SIN2(X)
*     anstelle des sin basteln wir die Funktion  2*sin
        SIN2 = 2.0*SIN(X)
      END

      SUBROUTINE PLOT (Bezeichnung, F , Nr_Z_Satz, neualt)

* plottet jede beliebige Funktion f(x)
* im Bereich 0 ó x ó 7.5  und  -2.5 ó y ó 2.5; Schrittweite 0.1 und 0.2
* Auflösung in X-Richtung:  ñ 0.05  (Zeichenbreite = 1 Schritt = 0.1)
* Auflösung in Y-Richtung:  ñ 0.033 (3 Zeichen _-â = 1 Schritt = 0.2)
* Der TEXT-Bildschirm (!!) wird auf eine Auflösung von 80 x 75 gebracht

* Aufruf: CALL PLOT ( Bezeichnung, Funktion, Zeichensatz, neualt)
*         Bezeichnung: max. 12 Character lang
*         Funktion: Fortran-Funktion oder selbst geschriebene Funktion
*                   im rufenden Programm muß dann stehen:
*                   INTRINSIC funktion    bzw. EXTERNAL funktion
*                   Beispiel s.o.
*         Zeichensatz: 1: dünn; 2: dick
*         neualt:  .TRUE. : Bild wird neu geschrieben
*                 .FALSE. : Bild wird in das alte eingetragen
*                           evtl. mit anderem Zeichensatz!

* Beschreibung der Parameter:
*               Bezeichnung hat Länge wie im rufenden Programm
      CHARACTER Bezeichnung*(*)
      REAL F
*     external f im rufenden Programm
      INTEGER Nr_Z_Satz
      LOGICAL neualt
```

```
* Beschreibung der lokalen Variablen:
*   X / Y - Werte der Funktion F
      REAL XWert, YWert
*   Ausgabe auf Gerät 6 (Bildschirm)
      INTEGER Ausgabe, i,ix
      PARAMETER (Ausgabe =6)
*   3 Zeichensätze: dünn, mittel, dick (Nr_Z_Satz=1/2/3) je nach Wunsch
      CHARACTER Zeichensatz ( 1:3, -1:+1)
*     wir numerieren die Zeilen von -12 bis +12 (25 Stück)
*                     die Spalten von -4 bis +76 (80 Stück)
      CHARACTER Seite(-4:+75, -12:+12),Zeilen ( -12:+12) *80, Symbol
*   Überlagerung Seite (einzelne Character) und Zeilen (je 80 Character)
      EQUIVALENCE(Seite, Zeilen)
*             Spalten / Zeilenindex für Seite, vorhergehende Zeile
      INTEGER Spalte, Zeile, Zeile_alt
*   Definition der Zeichen des dünnen (punktförmigen) ...
      Zeichensatz (1,-1) = CHAR( 46)
      Zeichensatz (1, 0) = CHAR(249)
      Zeichensatz (1,+1) = CHAR( 96)
*     Zeichensatz (1,+1) = CHAR( 39) falls Funktion steigt, s.u.
*   Definition der Zeichen des mittleren (strichförmigen) ...
      Zeichensatz (2,-1) = CHAR( 95)
      Zeichensatz (2, 0) = CHAR(196)
      Zeichensatz (2,+1) = CHAR(126)
*         ... und des dicken Zeichensatzes (ausgefüllte Felder)
      Zeichensatz (3,-1) = CHAR(220)
      Zeichensatz (3, 0) = CHAR(254)
      Zeichensatz (3,+1) = CHAR(223)
      DATA Zeilen/25*' '/
* Ende der Typdeklarationen und der Zuweisung von Anfangswerten
* falls neualt = .TRUE. , werden alle 25 Zeilen des Schirms gelöscht
      IF (neualt) THEN
         DO I=-12,+12
            Zeilen(I)=' '
         ENDDO
      ENDIF
* erstellen x/y-Achsen mit Beschriftung
*     x-Achse Sp. 0 bis 76 durchziehen
      DO Spalte= 0, 75
*              in Zeile 0
         Seite ( Spalte, 0) = CHAR(196)
      ENDDO
* Markierungen alle 10 Stellen
      DO Spalte =0, 75, 10
         Seite (Spalte,0) = CHAR(194)
*        darunter in Zeile 1 die Ziffern 1 bis 7
         IF(MOD(Spalte,10).EQ.0) WRITE (Seite(Spalte,1),'(I1)')Spalte/10
      ENDDO
* y-Achse in Zeile -12 bis +12
      DO Zeile =-12, +12
*       in Sp. 0
         Seite(0, Zeile) = CHAR(179)
      ENDDO
* Markierungen wie oben
      DO Zeile =-10, +10, 5
         Seite(0, Zeile) = CHAR(180)
*        davor in Sp. -2 die Ziffern 2,1,0,1,2
         WRITE (Seite( -2, Zeile), '(I1)') ABS(Zeile/5)
*        ein Minus für die beiden unteren Ziffern
         IF ( Zeile .GT.0 ) Seite(-3,Zeile)='-'
      ENDDO
```

```
* Berechnung der 76 Funktionswerte von 0 bis 7.5 in Schritten von 0.1
      Zeile_alt = 0
      DO ix = 0, 75
*        damit XWERT niemals unter den wahren Wert fällt
         XWert = ix*0.1000001
*        Aufruf der Funktion f
         YWert = F(XWert)
*        die Zeilen dehnen wir mit Faktor 5!
         Zeile =-NINT(YWert *5.0)
*                nächste ganze Zahl
* Umsetzung in Zeilen/Spaltenindex, da Zeilen/Spalten 1-10 leer bleiben
         Spalte =ix
*        prüfen, ob Indizes im richtigen Bereich
*        wenn ja, Punkt eintragen - obere oder untere Hälfte oder Mitte!
         IF (Zeile .GE. -12 .AND. Zeile  .LE.12
     &      .AND. Spalte .GE. -4 .AND. Spalte .LE. 75) THEN
*           wir tun jetzt so, als ob der Bildschirm 75 Zeilen hätte ...
            IF ( ABS( YWert*5.0 - NINT(YWert*5.0) ) .LE. 1.0/6.0) THEN
*              dann liegt YWert in der Mitte des 0.2- Intervalls
               Symbol = Zeichensatz ( Nr_Z_Satz, 0)
*              dieses Zeichen füllt die Mitte aus
            ELSEIF ( INT ( YWert*5.0 +0.5) .GT. INT(YWert*5.0)) THEN
*              dann liegt YWert in der unteren Hälfte des 0.2-Intervalls
               Symbol = Zeichensatz ( Nr_Z_Satz, -1)
*              dieses Zeichen füllt das untere Drittel aus
            ELSE
*              dann liegt YWert in der oberen Hälfte des 0.2- Intervalls
               Symbol = Zeichensatz ( Nr_Z_Satz, +1)
*              dieses Zeichen füllt das obere Drittel aus
               IF(Nr_Z_Satz.EQ.1 .AND. Zeile_alt.GE.Zeile) Symbol = ''''
*               ' statt ` falls Funktion steigt
            ENDIF
*        jetzt erst wird das entspr. Zeichen in die Tabelle eingetragen
         Seite(Spalte,Zeile)= Symbol
         ENDIF
         Zeile_alt = Zeile
      ENDDO
* Ausgabe des ganzen Plots: Überschrift, max. 16 Zeichen
      Zeilen(-12)  (1:16) = Bezeichnung
*     Drucken zeilenweise in Stücken von je 80 Zeichen
*     zunächst die ersten 24 Zeilen
      WRITE(UNIT=Ausgabe,FMT=*) ( Zeilen(I), I=-12,+11 )
*     das 80. Zeichen der letzten Zeile wird nicht gezeigt: der Cursor
*     bleibt dort stehen (Formatcode $), sodaß der Plot sichtbar bleibt.
      WRITE(UNIT=Ausgabe,FMT= '( A$ )' ) Zeilen(12) ( :79)
*     weiter mit Taste Enter, s.oben
      END
```

Anhang A: Kurzbeschreibung der Syntax von Fortran

Hier wird nur der im Skriptum behandelte Teil der Syntax beschrieben.
Angaben in **[...]** können weggelassen werden.
Anweisung (|): Block Kommentar

Hauptprogrammstruktur

```
  PROGRAM programmname                        Name des Hauptprogramms
   Deklarationen
   ausführbare Anweisungen
  END
```

Unterprogrammstruktur

```
  [ Typ ] FUNCTION name ( Parameter-Liste ) Kopfzeile der Funktion
   Deklarationen                            (Typ kann auch Character
   ausführbare Anweisungen                      sein)
  END

  SUBROUTINE Name [ ( Parameter-Liste ) ]   Kopfzeile der Subroutine
   Deklarationen
   ausführbare Anweisungen
  END
```

Deklarationen

```
  CHARACTER [ * Länge ] Deklarations-Liste  Fortran-Typen
  INTEGER Deklarations-Liste                        .
  REAL Deklarations-Liste                           .
  LOGICAL Deklarations-Liste                        .
  COMPLEX Deklarations-Liste                        .
  REAL*8 Deklarations-Liste                         .
  Typ Dimensions-Liste                      Feldvereinbarung
  DATA Variable / Werte-Liste / [ ... ]     Anfangswerte
  EXTERNAL Funktionsnamen-Liste             Funktionsnamen werden an
                                            ein Unterprogramm übergeben
```

Ausführbare Anweisungen

```
  READ(5,*) E/A-Liste                       formatfreie Eingabe
  READ*,    E/A-Liste                       dass. Kurzform
  WRITE(6,*) E/A-Liste                      formatfreie Ausgabe
  PRINT*,   E/A-Liste                       dass. Kurzform
  READ (UNIT=..,FMT=..) E/A-Liste           dass. mit expliziter
  WRITE (UNIT=..,FMT=..) E/A-Liste                Geräte-Nr. und Format
n FORMAT ( Formatcodes )                    n: 1 ... 99999 in Sp. 1-5

  Variable = Ausdruck                       Zuweisung; arithmetisch,
                                               logisch oder character
  IF ( logischer Ausdruck ) THEN          ( IF
       ausführbare Anweisungen              |
[ ELSEIF ( logischer Ausdruck ) THEN  ]    |
[    ausführbare Anweisungen          ]    |
[ ELSE                                ]    |
[    ausführbare Anweisungen          ]    |
  ENDIF                                 ) ENDIF

  IF ( log.Ausdruck ) ausführb.Anweisung    Kurzform des IF

     X = Anfangswert                        WHILE-Schleife
  WHILE(Bedingung)                        (   solange Bedingung wahr
     ausführbare Anweisungen              |
     IF (Endebedingung) GOTO e           -->evtl.Verlassen der Schleife
     ausführbare Anweisungen              |
     X = X + Schrittweite                 |   Veränderung der Bedingung
  ENDWHILE                                )
e CONTINUE                                    hier weiter falls   GOTO e
```

```
  DO,I=Anfangswert,Endwert[,Schrittweite]     ( DO-Schleife
        ausführbare Anweisungen                 |
  ENDDO                                        )
```

Implizite DO-Schleife in E/A-Liste:
z.B. READ, (A(I), I=Anfangswert,Endwert **[**,Schrittweite**]**)

Funktionsname (Argumenten-Liste) **Funktionsaufruf**
CALL Subroutine-Name **[** (Argumenten-Liste) **] Aufruf einer Subroutine**

Stichwort	**Kurzbeschreibung**
Name	Buchstabe plus bis zu 30 Buchstaben / Ziffern
Variable	Name bezeichnet eine einfache Variable oder ein Feld
Deklarations-Liste	Variable, wahlweise mit Länge, getrennt durch Komma
Dimensions-Liste	Feldname, wahlweise mit Länge, gefolgt von der Max-Dimension, getrennt durch Komma oder (von:bis) bzw. (von:bis,von:bis,...)
Feldname	Name (mit Dimensionsangabe im Deklarationsteil)
Max-Dimension	(n) oder (n,m) oder 3,4,5,6,7 Dimensionen
Werte-Liste (auch für READ)	Konstante,getrennt durch Komma;im Freiformat mehrere Konstante der Form Wiederholungsfaktor * Konstante
E/A-Liste	Variable, Konstante, Ausdrücke, getrennt durch Komma
E/A-Einheit	5: Bildschirm; 6: Schirm oder Drucker (0-99 möglich)
n in E/A-Anweisung	verweist auf das zugehörige FORMAT (*: Freiformat); steht in Sp. 1-5 der FORMAT-Anweisung
Bedingung	in der WHILE-Schleife: WHILE(Bedingung) läßt die Schleife laufen, solange die Bedingung erfüllt ist
GOTO e (Schleife)	Nummer 1. Anweisung hinter der letzten der Schleife 1. Schleifenanweisung springt dahin: IF(...) GOTO e
Formatcodes	autonom ablaufende: / nX Tn 'text' : /: neuer Record(Zeile); X: Leerzeichen; T: Tabulator beschreibende:wAz wGz.s für Strings bzw. andere Typen nur für den entspr. Typ: wIz wLz wFz.d wEz.d wDz.d z: Zahl Druckstellen; s: Zahl signifikanter Ziffern (G-Format), d: Zahl der Ziffern rechts vom . w: Wiederholungsfaktor; dazu Klammerung bis Tiefe 7
Variable	Name oder Feldelement
Feldelement	Name(Dimensionsangabe)
Ausdruck	arithmetischer oder logischer oder character-Ausdruck
arithm. Ausdruck	Variable / Konstante / numerische Funktionen mit arithmetischen Operatoren und Klammern
logischer Ausdruck	Variable / Konstante / logische Funktionen mit logischen oder Vergleichs-Operatoren und Klammern
Character-Ausdruck	Character -Variable/ -Konstante mit -Operator und ( )
String-Konstante	'...'; im Text wird ein ' doppelt als '' geschrieben
integer-Konstante	ganze Zahl wahlweise mit Vorzeichen
real-Konstante	Zahl mit Dezimalpunkt wahlweise mit Vorzeichen oder mit E oder D und Exponent (auch mit Vorzeichen) 1-7 Dezimalen, oder mit E Exponent (einfach genau), 8-16 Dezimalen und mit D Exponent (doppelt genau)
logical-Konstante	.TRUE. oder .FALSE. als logisch "wahr" oder "falsch"
Argumenten-Liste	Variable, Konstante, Feldelemente, Feldnamen aus dem rufenden Programm:die entspr. Werte werden übergeben Mit EXTERNAL: die entspr. Namen werden übergeben
Parameter-Liste	Variable, Konstante, Feldelemente, Feldnamen als Platzhalter für die Argumente des rufenden Programms
arithm. Operator	+ - * / ** für Add.,Subtr.,Mult.,Div.,Exponentiation
Vergleichs-Operator	.EQ. .NE. .LT. .GT. .LE. .GE. für = /= < > <= >=
logischer Operator	.NOT. .OR. .AND. als logisches "nicht" "oder" "und"
Character-Operator	Substring: (von:bis) erzeugt entspr. Teilstring Konkatenierung: // verkettet 2 Strings zu einem

Anhang B: Literatur zu Fortran

Ziehen Sie nur Bücher in Betracht, in denen ausdrücklich der neuere Fortran-77-Standard beschrieben ist!

Ein besonders gutes Preis-Leistungsverhältnis weisen die folgenden deutschsprachigen Einführungen auf (Signatur der URZ-Bibliothek): (Preise Stand: 1986)

	Signatur
I.Kiessling, M.Lowes: Programmierung mit Fortran 77, 4. Aufl. 1987, Teubner, DM 16.80	FP8202
E.Kaucher et al: Programmiersprachen im Griff: Band 5: Fortran 77, BI, DM 29,80	FP8303
E.Kaucher, et al. Programmiersprachen im Griff Band 7: Übungen und Tests in Fortran 77	FP8402
G. May: Strukturiertes Programmieren mit Fortran Hanser, DM 34.00	FP8002

Für Fortgeschrittene empfehlenswert:

J.L. Wagener: Fortran 77: Principles of Programming John Wiley, (englisch) 61.90	FP8006

Als Standard-Referenz:

Harry Katzan: Fortran 77, Van Nostrand Reinhold Comp. (englisch), DM 88.40	FP7802

Sehr empfehlenswert als Einführung in Fortran und darüber hinaus in Programmentwicklung und Anwenden von Software:

Daniel D.McCracken: Computerpraxis mit Fortran77 in Naturwissenschaft und Technik, Hanser Verlag 1985, DM 68.-	FP8505

Programmierstil (mit Ausblick auf den Fortran90-Standard):

Kessel, Stephen R.: Fortran77 Dokumentation and Style, Addison & Wesley (englisch)	FP9201

Anhang C: Der WATFOR Debugger

Start: **RUN/DE**bug

Sie erhalten die Zeilennummer und die erste ausführbare Anweisung.

DEBUG> fordert Sie auf, ein Debugger-Kommando einzugeben, z.B.

Help gibt Ihnen eine kurze Übersicht über die Kommandos.
Anstelle von *isn* (Internal Statement Number) kann stehen:
. / *n* die momentan angezeigte Anweisung / Anweisung Nr. *n*
+*n* / -*n* *n* Anweisungen vorwärts / rückwärts

Display listet +- 5 Anweisungen (insgesamt 10)
Display *isn* listet Anweisung Nr. *isn*
Display *isn*:*isn* listet Anweisungen von:bis
Display * listet das ganze Programm

Go startet die Programmausführung bei der momentanen isn
Go *isn* startet die Programmausführung bei *isn*

zeigt die E/A-Geräte an, die geloggt werden
LOGio *unit* **listet ab sofort alle Ein- und Ausgabe betreffs** *unit*
im Listing bedeutet: < : Eingabe; > : Ausgabe
LOGio OFF schaltet aus

Quit beendet den Debugger, landet im Editor

SET *item value* **weist der Variablen (Substring)** *item* **den Wert** *value* zu

STAtus zeigt die eingeschalteten Optionen (STOp, Trace, Step)

STEp Stop bei jeder Anweisung; weiter mit der Enter-Taste
STEp OVER dass. ohne Stop bei CALL (Ende Einzelschritt mit Go)

STOp *isn* **wie Go, stoppt aber bei** *isn*
STOp *isn* : *isn* dass., stoppt aber bei allen *isn* im Bereich von:bis
STOp Calls dass., stoppt aber nur bei CALL (Unterprogrammaufruf)
STOp OFF schaltet ab

SYStem geht kurz ins DOS (für DOS-Kommandos); EXIT geht zurück
SYStem *command* führt das DOS-Kommando *command* aus

Trace schaltet Anzeige aller ausgeführten Anweisungen ein
Trace *isn* dasselbe nur für die Anweisung Nr. *isn*
Trace *isn* : *isn* dasselbe nur für die Anweisungen von : bis
Trace Calls dasselbe, zeigt aber nur die Unterprogrammnamen
Trace OFF schaltet ab

Unit zeigt alle offenen E/A-Geräte und File-Attribute
Unit *unit* dasselbe für Gerät *unit* (0...99)

Where Traceback (rufende Programme) und nächste Anweisung

? zeigt alle Variablen und ihre Werte; Halt mit Strg_Untbr
? *fmt var*,... zeigt die Werte der entspr. Variablen; Format kann sein
(kann wegbleiben): (A) : A-Format; (Z) : Hexadezimal

Nothalt im laufenden Programm: "Strg_Untbr" (PC), Wirkung wie "Where",
danach sind bel. Debug-Kommandos möglich

Anhang D: Lösungen zu den Übungsaufgaben

```
PROGRAM Kapitel_1_Aufgabe_1
INTEGER Summe
Summe = 127 + 584
PRINT*,'Die Summe von 127 und 584 ist',Summe
END

PROGRAM Kapitel_1_Aufgabe_2
PRINT*,' *****  *   *  *   *   *** '
PRINT*,' *      ** **  ** **  *   *'
PRINT*,' *      * * *  * * *  *   *'
PRINT*,' *****  *   *  *   *  *****'
PRINT*,' *      *   *  *   *  *   *'
PRINT*,' *****  *   *  *   *  *   *'
END

PROGRAM Kapitel_1_Aufgabe_3
INTEGER Zahl
PRINT*,'Quadrate und Kuben der Zahlen von 1 bis 20:'
PRINT*,'        Zahl     Quadrat        Kube'
Zahl = 1
WHILE( Zahl .LE. 20)
  PRINT*,Zahl, Zahl*Zahl,Zahl*Zahl*Zahl
  Zahl = Zahl + 1
ENDWHILE
END

PROGRAM Kapitel_1_Aufgabe_4
INTEGER Zahl
PRINT*,'Quadrate und Kuben der ungeraden Zahlen von 1 bis 20:'
PRINT*,'        Zahl     Quadrat       Kubus'
Zahl = 1
WHILE( Zahl .LE. 20)
  PRINT*,Zahl, Zahl*Zahl, Zahl*Zahl*Zahl
  Zahl = Zahl + 2
ENDWHILE
END

PROGRAM Kapitel_1_Aufgabe_5
INTEGER Lohn, Tag
Lohn = 1
Tag = 1
WHILE( Tag .LE. 30)
  Lohn = 2*Lohn
  Tag = Tag + 1
ENDWHILE
PRINT*,'Nach 30 Tagen beträgt der Lohn',Lohn,' Pfennige'
PRINT*,'Das sind', Lohn/100,' Mark  und',Lohn-Lohn/100*100,
& ' Pfennige'
END

PROGRAM Kapitel_1_Aufgabe_6
INTEGER Jahr
REAL Schuld, Zinsen
Schuld = 1000
Jahr = 1991
```

```
WHILE( Schuld .GT. 0)
  Zinsen = 0.10*Schuld
  Schuld = Schuld + Zinsen - 200
  PRINT*,'Schuld am Ende des Jahres',Jahr,':',Schuld,' Mark'
  PRINT*,'  An Zinsen müssen',Zinsen,' bezahlt werden'
  Jahr = Jahr + 1
ENDWHILE
PRINT*,'Endlich schuldenfrei im Jahr',Jahr
PRINT*,'Guthaben:',-Schuld,' Mark'
END
```

```
PROGRAM Kapitel_1_Aufgabe_7
REAL x,y
x = 0
WHILE( x .LE. 10 )
  y = ( x*x - x + 2 ) / ( x + 3 )
  PRINT*,'Für x =',x,'     wird y =',y
  x = x + 1
ENDWHILE
END
```

```
PROGRAM Kapitel_1_Aufgabe_8
REAL x,y
x = 1
WHILE( x .LE. 21 )
  y = 3*x*x - 2*x + 7
  PRINT*,'Für x =',x,'     wird y =',y
  x = x + 2
ENDWHILE
END
```

```
PROGRAM Kapitel_1_Aufgabe_9
INTEGER Zahl
REAL Summe
Summe = 0
Zahl = 0
WHILE( Zahl .LE. 9 )
  Summe = Summe + Zahl**3
  PRINT*,'Wurzel der Summe der Kuben von 0 bis',Zahl,' ist',
&  SQRT(Summe)
  Zahl = Zahl + 1
ENDWHILE
PRINT*,'Die Ergebnisse unterscheiden sich jeweils um 1,2,3, usw.'
END
```

```
PROGRAM Kapitel_1_Aufgabe_10
INTEGER Zahl1, Zahl2, Summe, Nummer
  PRINT*,'Fibonacci-Zahl Nummer           0 ist', 0
  PRINT*,'Fibonacci-Zahl Nummer           1 ist', 1
Zahl1 = 0
Zahl2 = 1
Nummer = 2
WHILE( Nummer .LE. 15 )
  Summe = Zahl1 + Zahl2
  PRINT*,'Fibonacci-Zahl Nummer',Nummer,' ist', Summe
  Zahl1 = Zahl2
  Zahl2 = Summe
  Nummer = Nummer + 1
ENDWHILE
END
```

```
      PROGRAM Kapitel_1_Aufgabe_11
      REAL x
      INTEGER Exponent, Zaehler
      x = 1.25
      Exponent = 1
      Zaehler = 1
      PRINT*,'x=',x
      WHILE( Zaehler .LE. 5 )
        PRINT*,'x hoch',Exponent,' ist', x**Exponent
        Exponent = 2 * Exponent
        Zaehler = Zaehler + 1
      ENDWHILE
      END

      PROGRAM Kapitel_1_Aufgabe_12
      REAL x
      INTEGER Exponent, Nenner, Differenz, Zaehler
      x = 1.25
      Exponent = 0
      Differenz = 1
      Zaehler = 1
      PRINT*,'x=',x
      WHILE( Zaehler .LE. 8 )
        Exponent = Exponent + Differenz
        Nenner = Exponent
        PRINT*,'x hoch',Exponent,' durch',Nenner,' ist',
     &          x **    Exponent   /      Nenner
        Differenz = Differenz + 1
        Zaehler = Zaehler + 1
      ENDWHILE
      END

      PROGRAM Kapitel_2_Aufgabe_1
      IMPLICIT NONE
      INTEGER i
      i = 1
      WHILE( i .LE. 12 )
        PRINT*, i, 12 - ( 6 - i )
        i = i + 1
      ENDWHILE
      END

      PROGRAM Kapitel_2_Aufgabe_2
      IMPLICIT NONE
      REAL x,y
      x = 0.0
      WHILE( x .LE. 2.0 )
        y = ( x**3 - 4*x**2 + x - 3 ) / ABS ( x + 2 )
        PRINT*,'für x=',x,' ist y=',y
        x = x + 0.25
      ENDWHILE
* um die Ausgabe auf den Schirm anzuhalten
      PRINT*,'weiter mit Enter'
      READ*
      x = -1.0
      WHILE( x .GE. -13.0 )
        y = ( x**3 - 4*x**2 + x - 3 ) / ABS ( x + 2 )
        PRINT*,'für x=',x,' ist y=',y
        x = x - 2.0
      ENDWHILE
```

```
* um die Ausgabe auf den Schirm anzuhalten
      PRINT*,'weiter mit Enter'
      READ*
      x = 10.0
      WHILE( x .GE. -1.0 )
        y = ( x**3 - 4*x**2 + x - 3 ) / ABS ( x + 2 )
        PRINT*,'für x=',x,' ist y=',y
        x = x - 1.0
      ENDWHILE
      PRINT*,'weiter mit Enter'
      END

      PROGRAM Kapitel_2_Aufgabe_3
      IMPLICIT NONE
      INTEGER n,Exponent
      REAL x
      PRINT*,'bitte n eingeben'
      READ*,n
      PRINT*,'bitte x eingeben'
      READ*,x
      Exponent = 0
      WHILE( Exponent .LE. n )
        PRINT*,'x /',2**Exponent,' ist',x / 2**Exponent
        Exponent = Exponent + 1
      ENDWHILE
      END

      PROGRAM Kapitel_2_Aufgabe_4
      IMPLICIT NONE
      INTEGER n
      n = 20
      WHILE( n .LE. 20 )
        PRINT*,'Summe der ersten',n,' Zahlen N    ist',
     &              n*(n+1)/2
        PRINT*,'Summe der ersten',n,' Zahlen N**2 ist',
     &              n*(n+1)*(2*n+1)/6
        PRINT*,'Summe der ersten',n,' Zahlen N**3 ist',
     &              n**2*(n+1)**2/4
        PRINT*,'Summe der ersten',n,' Zahlen N**4 ist',
     &              n*(n+1)*(2*n+1)*(3*n**2+3*n-1)/30
        PRINT*,'Summe der ersten',n,' Zahlen N**5 ist',
     &              n**2*(n+1)**2*(2*n**2+2*n-1)/12
        n = n + 1
      ENDWHILE
      END

      PROGRAM Kapitel_2_Aufgabe_5
      IMPLICIT NONE
      REAL x, Naeherung
      x = 0.5
      WHILE( x .LE. 9.5 )
        Naeherung = (1+4*x) / (4+x)
        PRINT*,'Für x=',x,' ergibt sich als Näherung:',Naeherung
        PRINT*,'Der prozentualer Fehler beträgt',
     &              ( Naeherung - SQRT(x) ) / SQRT(x) *100,' %'
        IF ( x .EQ. 5.0 ) THEN
* um die Ausgabe auf den Schirm anzuhalten
          PRINT*,'weiter mit Enter'
          READ*
        ENDIF
        x = x + 0.5
      ENDWHILE
      END
```

```
      PROGRAM Kapitel_2_Aufgabe_6
      IMPLICIT NONE
      REAL x, Naeherung
      x = 0.2
      WHILE( x .LE. 1.0 )
        Naeherung = -0.076 + 0.281*x - 0.238 / ( x + 0.15 )
        PRINT*,'Für x=',x,' ergibt sich als Näherung:',Naeherung
        PRINT*,'Der absolute Fehler beträgt',
     &         ABS ( Naeherung - LOG10(x) )
        x = x + 0.2
      ENDWHILE
      END

      PROGRAM Kapitel_2_Aufgabe_7
      IMPLICIT NONE
      INTEGER Summe,Rest
      INTEGER Tausender, Fuenfhunderter, Zweihunderter, Hunderter,
     &    Fuenfziger, Zwanziger, Zehner, Fuenfer
      Summe = 1575
* Tausender
      Tausender = Summe / 1000
      Rest = Summe - Tausender *  1000
* Fünfhunderter
      Fuenfhunderter = Rest / 500
      Rest = Rest - Fuenfhunderter *  500
* Zweihunderter
      Zweihunderter = Rest / 200
      Rest = Rest - Zweihunderter *  200
* Hunderter
      Hunderter = Rest / 100
      Rest = Rest - Hunderter *  100
* Fünfziger
      Fuenfziger = Rest / 50
      Rest = Rest - Fuenfziger *  50
* Zwanziger
      Zwanziger = Rest / 20
      Rest = Rest - Zwanziger *  20
* Zehner
      Zehner = Rest / 10
      Rest = Rest - Zehner *  10
* Fünfer
      Fuenfer = Rest / 5
      Rest = Rest - Fuenfer *  5
      PRINT*,'Man braucht für',Summe,' mindestens an Scheinen:'
      PRINT*,Tausender,' Tausender'
      PRINT*,Fuenfhunderter,' Fünfhunderter'
      PRINT*,Zweihunderter,' Zweihunderter'
      PRINT*,Hunderter,' Hunderter'
      PRINT*,Fuenfziger,' Fünfziger'
      PRINT*,Zwanziger,' Zwanziger'
      PRINT*,Zehner,' Zehner'
      PRINT*,Fuenfer,' Fünfer'
      END

      PROGRAM Kapitel_3_Aufgabe_1
      IMPLICIT NONE
      INTEGER Zahl, Zaehler, Ziffer, Expo
      PRINT*,'bitte Zahl eingeben'
      READ*,Zahl
      Zaehler = 0
      Expo = 10
```

```
      WHILE ( Expo .GE. 0 )
        Ziffer = Zahl / 10**Expo
        IF( Ziffer .NE. 0 ) THEN
            Zaehler = Zaehler + 1
            Zahl = Zahl - Ziffer*10**Expo
        ENDIF
        Expo = Expo - 1
      ENDWHILE
      PRINT*,'Zahl der Ziffern ungleich 0:',Zaehler
      END

      PROGRAM Kapitel_3_Aufgabe_2
      IMPLICIT NONE
      INTEGER n,m,k, nk,mk
      PRINT*,'bitte 3 Zahlen n,m,k eingeben'
      READ*,n,m,k
      nk = MOD(n,10**k)
      mk = MOD(m,10**k)
      IF( nk .EQ. mk ) THEN
       PRINT*,'die letzten',k,' Stellen von',n,' und',m,' sind gleich'
      ELSE
       PRINT*,'die letzten',k,' Stellen von',n,' und',m,' sind ungleich'
      ENDIF
      END

      PROGRAM Kapitel_3_Aufgabe_3
      IMPLICIT NONE
      INTEGER a,b,c
      PRINT*,'bitte 3 Seitenlängen eingeben'
      READ*,a,b,c
      IF( a .GT. b+c .OR. b .GT. a+c .OR. c .GT. a+b
     & .OR. a .EQ. 0 .OR. b .EQ. 0 .OR. c .EQ. 0 ) THEN
        PRINT*,'das gibt kein Dreieck'
        PRINT*,0
      ELSEIF( a .NE. b .AND. a .NE. c .AND. b .NE. c ) THEN
        PRINT*,'alle Seiten ungleich'
        PRINT*,1
      ELSEIF( a .EQ. b .AND. b .EQ. c .AND. c .EQ. a ) THEN
        PRINT*,'alle Seiten gleich'
        PRINT*,3
      ELSEIF( a .EQ. b .OR. a .EQ. c .OR. b .EQ. c ) THEN
        PRINT*,'2 Seiten gleich'
        PRINT*,2
      ENDIF
      END
      PROGRAM Kapitel_3_Aufgabe_4
      IMPLICIT NONE
      REAL Zahl, low, high, mid
      Zahl = 12.0
      low = 3.0
      high = 4.0
      WHILE( high - low .GE. 0.02 )
        PRINT*,'low - high:',low,high
        mid = ( low + high ) / 2.0
        IF ( mid**2 .GT. Zahl ) THEN
          high = mid
        ELSE
          low = mid
        ENDIF
      ENDWHILE
      PRINT*,'Näherung für Wurzel aus 12:',mid
      PRINT*,'SQRT(12)=                  ',SQRT(12.0)
      END
```

```
PROGRAM Kapitel_3_Aufgabe_5
IMPLICIT NONE
INTEGER Jahr,Monat,Tag,  J,M,T,  Wochentag,K
PRINT*,'Berechnung des Wochentags aus dem Datum'
WHILE ( Jahr .GE. 0 )
  PRINT*
  PRINT*,'bitte Jahr eingeben'
  READ*,Jahr
  PRINT*,'bitte Monat eingeben'
  READ*,Monat
  PRINT*,'bitte Tag eingeben'
  READ*,Tag
  IF ( Monat .EQ. 1 .OR. Monat .EQ. 2 ) THEN
    M = Monat + 12
    J = Jahr - 1
  ELSE
    J = Jahr
    M = Monat
  ENDIF
  T = Tag
  K = T + 2*M + (3*M +3) / 5 + J + J/4 + J/100 + 1
  Wochentag = MOD(K,7)
  IF ( Wochentag .EQ. 1 ) THEN
    PRINT*,'Der Tag ist ein Montag'
  ELSEIF ( Wochentag .EQ. 2 ) THEN
    PRINT*,'Der Tag ist ein Dienstag'
  ELSEIF ( Wochentag .EQ. 3 ) THEN
    PRINT*,'Der Tag ist ein Mittwoch'
  ELSEIF ( Wochentag .EQ. 4 ) THEN
    PRINT*,'Der Tag ist ein Donnerstag'
  ELSEIF ( Wochentag .EQ. 5 ) THEN
    PRINT*,'Der Tag ist ein Freitag'
  ELSEIF ( Wochentag .EQ. 6 ) THEN
    PRINT*,'Der Tag ist ein Samstag'
  ELSE
    PRINT*,'Der Tag ist ein Sonntag'
  ENDIF
ENDWHILE
PRINT*,'Stop nach negativem Jahr'
END
PROGRAM Kapitel_3_Aufgabe_6
IMPLICIT NONE
INTEGER Zahl, Ziffer1, Ziffer2, Ziffer3
INTEGER Zahl1, Zahl2, Zahl3, Zahl4, Zahlen, Zeilen
Zahl   = 100
Zahlen = 0
Zeilen = 0
WHILE ( Zahl .LE. 500 )
  Ziffer1 = Zahl / 100
  Ziffer2 = MOD(Zahl,100) / 10
  Ziffer3 = MOD(Zahl,10)
  IF(Ziffer1 .NE. Ziffer2 .AND. Ziffer2 .NE. Ziffer3 .AND.
&     Ziffer3 .NE. Ziffer1 ) THEN
    Zahlen = Zahlen + 1
    IF ( Zahlen .EQ. 1 ) THEN
      Zahl1 = Zahl
    ELSEIF ( Zahlen .EQ. 2 ) THEN
      Zahl2 = Zahl
    ELSEIF ( Zahlen .EQ. 3 ) THEN
      Zahl3 = Zahl
    ELSEIF ( Zahlen .EQ. 4 ) THEN
      Zahl4 = Zahl
  ENDIF
```

```
        IF( Zahlen .EQ. 4 ) THEN
          Zeilen = Zeilen + 1
          PRINT*,Zahl1,Zahl2,Zahl3,Zahl4
          Zahlen = 0
          IF( Zeilen .EQ. 23 ) THEN
            Zeilen = 0
            PRINT*,'weiter mit Enter'
            READ*
        ENDIF
        Zahl = Zahl + 1
      ENDWHILE
      END
```

```
      PROGRAM Kapitel_4_Aufgabe_1
      IMPLICIT NONE
      REAL x, y
      INTEGER I
      DO I= 1,20
        x = I
        y = (x**2 - x + 7) / (x**2 + 2)
        PRINT*,'für x=',x,' ist y=',y
      ENDDO
      PRINT*,'weiter mit Enter'
      READ*
      DO I= 3,24,3
        x = I
        y = (x**2 - x + 7) / (x**2 + 2)
        PRINT*,'für x=',x,' ist y=',y
      ENDDO
      PRINT*,'weiter mit Enter'
      READ*
      DO I= 0,-8,-1
        x = I
        y = (x**2 - x + 7) / (x**2 + 2)
        PRINT*,'für x=',x,' ist y=',y
      ENDDO
      PRINT*,'weiter mit Enter'
      READ*
      DO I= 12,-10,-2
        x = I
        y = (x**2 - x + 7) / (x**2 + 2)
        PRINT*,'für x=',x,' ist y=',y
      ENDDO
      END
```

```
      PROGRAM Kapitel_4_Aufgabe_2
      IMPLICIT NONE
      REAL x, y, Nenner
      INTEGER I
      DO I= -100,100,25
        x = I*0.01
        Nenner = 16.0*x**2 + 8*x - 3
* Nenner 0 ?
        IF ( ABS( Nenner ) .LT. 0.00001 ) THEN
* ja
          PRINT*,'Für x=',x,' wird der Nenner 0'
        ELSE
* nein: Normalfall
          y = (  3.0*SIN(x)**3 + 4.0*COS(x)**3  )  /  Nenner
          PRINT*,'Für x=',x,' wird y=',y
        ENDIF
      ENDDO
      END
```

```
      PROGRAM Kapitel_4_Aufgabe_3
      IMPLICIT NONE
      REAL x, x0, Kandidat
      INTEGER I
      Kandidat = 2.0
* eine erste Näherung für 1 / Wurzel aus 2
      x0 = 0.4
      DO I = 1,5
* x ist die verbesserte Näherung
        x = 0.5 * x0 * ( 3.0 - Kandidat * x0**2 )
* aus neu mach alt
        x0 = x
      ENDDO
      PRINT*,'nach 5 Iterationen ist die Näherung für 1 / Wurzel 2:',x
      PRINT*,'Zur Kontrolle: 1 / SQRT(2)=',1.0 / SQRT(Kandidat)
      END

      PROGRAM Kapitel_4_Aufgabe_4
      IMPLICIT NONE
      REAL x, x0, A
      INTEGER I
      A = 3.0
* eine erste Näherung für 1 / A
      x0 = 0.5
      DO I = 1,5
* x ist die verbesserte Näherung
        x = x0 * ( 2.0 - A * x0 )
        PRINT*,'Iteration Nr.',I,' gibt x=',x
* aus neu mach alt
        x0 = x
      ENDDO
      PRINT*,'nach 5 Iterationen ist die Näherung für 1 /',A,x
      PRINT*,'Zur Kontrolle: 1 /', A ,' =',1.0 / A
      END

      PROGRAM Kapitel_4_Aufgabe_5
* Aufgabenstellung:
*   Berechnung von n Fibonacci-Zahlen (hier n=15)
*   Definition der Fibonacci-Zahlen:
*       gegeben sind 2 ganze Zahlen als Anfangswerte;
*       3. Zahl = 2. + 1.; 4. = 3. + 2.; usw.
*   Nr.1 und 2 werden eingelesen (inneres WHILE mit nr als Zaehler)
*   dies wird bel. wiederholt (äußeres WHILE(.TRUE.) mit l1 als Zähler
      IMPLICIT NONE
      INTEGER A,B,C, D,NR, N
      N=15
      D=0
      WHILE(.TRUE.)
        PRINT*,'bitte 1. und 2. Fibonacci-Zahl eingeben; Ende: Strg_z'
        READ*,A,B
*       AT END,QUIT
*       ohne diese Anweisung gibt Strg_z einen  Error - end of file
        D=D+1
        PRINT*,'DURCHGANG NR.',D
        NR=1
* die ersten 2 Zahlen werden gesondert ausgegeben
        PRINT*,'Fibonacci-Zahl Nr.',NR,A
        NR=NR+1
        PRINT*,'Fibonacci-Zahl Nr.',NR,B
        NR=NR+1
* AB NR=3 läuft die WHILE-Schleife
```

```
        WHILE( NR .LE. N )
* Zahl c ist Summe der beiden vohergehenden (b,a)
          C=A+B
          PRINT*,'Fibonacci-Zahl Nr.',nr,c
* b rückt an den Platz von a
          A=B
* c rückt an den Platz von b
          B=C
          NR=NR+1
        ENDWHILE
        PRINT*,'Ende des Durchlaufs von',n,' Fibonacci-Zahlen'
      ENDWHILE
      PRINT*,'Ende des Programms nach',d,' Durchgängen'
      END
```

```
      PROGRAM Kapitel_4_Aufgabe_6
*  Erweiterung von Aufg. 5:
*    bei jedem Durchlauf 1...n werden die Quotienten je 2
*    aufeinander folgender Zahlen berechnet (Typ REAL!)
      IMPLICIT NONE
      INTEGER A,B,C, D,NR, N
      N=15
      D=0
        WHILE(.TRUE.)
        PRINT*,'bitte 1. und 2. Fibonacci-Zahl eingeben'
        READ*,A,B
        D=D+1
        PRINT*,'Durchgang Nr.',d
        NR=2
        WHILE(.TRUE.)
          IF(NR.GT.N) QUIT
          C=A+B
          PRINT*,'Quotient: Nr.',nr,'/ Nr.',nr-1,REAL(c)/REAL(b)
          A=B
          B=C
          NR=NR+1
        ENDWHILE
        PRINT*,'Ende eines Durchlaufs von',n,' Fibonacci-Zahlen'
      ENDWHILE
      PRINT*,'Ende des Programms nach',d,' Durchgängen'
      END
```

```
      PROGRAM Kapitel_4_Aufgabe_7
      IMPLICIT NONE
      REAL pn, pn_1, pn_2
      INTEGER n,Max_Paare
      Max_Paare = 10
* setzen Anfangswerte p0, p1
      pn_2 = 0
      pn_1 = 0.5
      PRINT*,'Die Wahrscheinlichkeit, daß keiner seinen Partner zieht'
        PRINT*,'ist bei',1,' teilnehmenden Paar  :',0
        PRINT*,'ist bei',2,' teilnehmenden Paaren:',0.5
      DO n = 3, Max_Paare
* x ist die verbesserte Näherung
        pn =  (n-1.0)/n * pn_1 + (1.0/n) * pn_2
        PRINT*,'ist bei',n,' teilnehmenden Paaren:',pn
* aus neu mach alt
        pn_2 = pn_1
        pn_1 = pn
      ENDDO
      PRINT*,'Viel vergnügen bei der Party'
      END
```

```
      PROGRAM Kapitel_4_Aufgabe_8
      IMPLICIT NONE
      INTEGER n, i,j,k, np,nn
      PRINT*,'bitte n eingeben (z.B. 5)'
      READ*,n
      PRINT*,'Tripel positiver Zahlen:'
      np = 0
      DO i=1,n
        DO j=1,n
          DO k=1,n
* hier werden alle Tripel von Zahlen von 1 bis n erzeugt
            IF ( i+j+k .EQ. n ) THEN
              np = np+1
              PRINT*,'Tripel Nr.',np,' ist',i,j,k
            ENDIF
      ENDDO
      PRINT*,'Tripel nicht negativer Zahlen:'
      PRINT*,'weiter mit Enter'
      READ*
      nn = 0
      DO i=0,n
        DO j=0,n
          DO k=0,n
* hier werden alle Tripel von Zahlen von 0 bis n erzeugt
            IF ( i+j+k .EQ. n ) THEN
              nn = nn+1
              PRINT*,'Tripel Nr.',nn,' ist',i,j,k
            ENDIF
      ENDDO
      PRINT*,'Insgesamt wurden',np + nn,' Tripel gefunden'
      END

      PROGRAM Kapitel_4_Aufgabe_9
      IMPLICIT NONE
      INTEGER i,j
      REAL x,y,z
      PRINT*,'z = (x**2 - 2*x*y + 3*y**3 ) / ( x*y + 3*x * 2*y + 4 ):'
      DO i=1,6
        x = i
        DO j=1,5
          y = j
          z = (x**2 - 2*x*y + 3*y**3 ) / ( x*y + 3*x * 2*y + 4 )
          PRINT*,'x=',x,' y=',y,' z=',z
        ENDDO
        PRINT*,'Weiter mit Enter'
        READ*
      ENDDO
      END

      PROGRAM Kapitel_4_Aufgabe_10
      IMPLICIT NONE
      INTEGER A,B, AA,BB, Summe
      A = 37
      B = 65
*     PRINT*,'bitte 2 Zahlen eingeben'
*     READ*,a,b
      AA = A
      BB = B
      Summe = B
```

```
      WHILE ( AA .GT. 0 )
        AA = AA / 2
        BB = BB * 2
        IF ( MOD ( REAL(AA),2.0) .NE. 0 ) Summe = Summe + BB
      ENDWHILE
      PRINT*,'Das Produkt von',A,' und',B,' ist',Summe
      PRINT*,'Benutzt wurde nur die Division / Multiplikation mit 2 !!'
      END

      PROGRAM Kapitel_4_Aufgabe_11
      IMPLICIT NONE
      LOGICAL Tom, Dick, Zack, Frage1, Frage2, Frage3
      INTEGER i,j,k
      Tom  = .true.
      Dick = .true.
      Zack = .true.
      DO i=1,2
        DO j=1,2
          DO k=1,2
           Frage1 = Tom .EQV. Dick
           Frage2 = Dick .EQV. (Tom .EQV. Zack)
           Frage3 = Zack .EQV. Dick
           IF( Frage1 .AND. .NOT. Frage2 .AND. Frage3 )
     &      PRINT*,'Jetzt wird klar: Tom,Dick,Zack sind ',Tom,Dick,Zack
           Tom = .false.
          ENDDO
          Dick = .false.
        ENDDO
        Zack = .false.
      ENDDO
      END

      PROGRAM Kapitel_4_Aufgabe_12
      IMPLICIT NONE
      REAL x, E
      INTEGER I
*     PRINT*,'Bitte Zahl eingeben'
*     READ*,x
      x = 4059.26
      PRINT*,'x=',x
      E = x * 0.5
      DO I=1,8
        E = E - ( E**2 -x ) / ( 2*E )
        PRINT*,'Näherung Nr.',I,' :',E
      ENDDO
      PRINT*,'Zur Kontrolle: SQRT(x)=   ',SQRT(x)
      END

      PROGRAM Kapitel_4_Aufgabe_13
      IMPLICIT NONE
      REAL x, Eneu, Ealt, Genauigkeit
      INTEGER I, Max_Iter
*     PRINT*,'Bitte Zahl eingeben'
*     READ*,x
      x = 3.0
      PRINT*,'x=',x
* setzen der Parameter für das Verfahren
      Max_Iter = 3
      Genauigkeit = 0.0001
* setzen 1. Näherung
      Ealt = x * 0.5
```

```
* Algorithmus
      DO I=1,Max_Iter
        Eneu = Ealt - ( Ealt**2 -x ) / ( 2*Ealt )
        PRINT*,'Näherung Nr.',I,' :',Eneu
* Abbruchkriterium
        IF ( ABS( Eneu - Ealt ) .LT. Genauigkeit ) QUIT
* für den neuen Iterationsschritt wird das "alte" Eneu zu Ealt
        Ealt = Eneu
      ENDDO
* ex post Analyse
      IF ( I .LE. Max_Iter ) THEN
        PRINT*,'Ende der Iteration mit Genauigkeit', Genauigkeit
      ELSE
        PRINT*,'Abbruch der Iteration nach',Max_Iter,' Schritten'
        PRINT*,'Genauigkeit', Genauigkeit,' wurde nicht erreicht'
      ENDIF
      PRINT*,'Zur Kontrolle: SQRT(x)=   ',SQRT(x)
      END
```

```
      PROGRAM Kapitel_5_Aufgabe_1
      IMPLICIT NONE
      INTEGER K(5), P(5), N(5), Z(5), i, np,nn,nz
      PRINT*,'bitte 5 Zahlen in einer Zeile eingeben'
      READ*,K
      np = 0
      nn = 0
      nz = 0
      DO i=1,5
        IF( K(i) .GT. 0 ) THEN
          np = np + 1
          P(np) = i
        ELSEIF( K(i) .LT. 0 ) THEN
          nn = nn + 1
          N(nn) = i
        ELSE
          nz = nz + 1
          Z(nz) = i
        ENDIF
      ENDDO
      PRINT*,'Werte:',K
      PRINT*,'Indizes der positiven Werte:',(P(i),i=1,np)
      PRINT*,'Indizes der negativen Werte:',(N(i),i=1,nn)
      PRINT*,'Indizes der zero      Werte:',(Z(i),i=1,nz)
      END
```

```
      PROGRAM Kapitel_5_Aufgabe_2
      IMPLICIT NONE
      INTEGER F(5), i ,j
      PRINT*,'bitte 5 Zahlen'
      READ*,F
      PRINT*,'Paare, deren Summe 12 ergibt:'
      DO i=1,4
        DO j=i+1,5
          IF (  F(i) + F(j) .EQ. 12 ) THEN
            PRINT*,F(i),F(j)
          ENDIF
      ENDDO
      END
```

```
      PROGRAM Kapitel_5_Aufgabe_3
      IMPLICIT NONE
      INTEGER Alter(10), i,j, Differenz(10,10),
     &        max_i,max_j,min_i,min_j, max_diff, min_diff
      PRINT*,'bitte 10 Alterszahlen'
      READ*,Alter
      DO i=1,9
        DO j=i+1,10
          Differenz ( i,j ) = ABS ( Alter(i) - Alter(j) )
      ENDDO
      Max_Diff = -1
      Min_Diff = 999999
      DO i=1,10
        DO j=i+1,10
          IF (Differenz ( i,j ) .GT. Max_Diff ) THEN
            Max_Diff = Differenz ( i,j )
            max_i = i
            max_j = j
          ELSEIF (Differenz ( i,j ) .LT. Min_Diff ) THEN
            Min_Diff = Differenz ( i,j )
            min_i = i
            min_j = j
          ENDIF
      ENDDO
      PRINT*,'Die Personen mit maximal verschiedenem Alter sind:'
      PRINT*,'Personen Nr.',max_i,max_j,' im Alter von',
     &        Alter(max_i),Alter(max_j)
      PRINT*,'Die Personen mit minimal verschiedenem Alter sind:'
      PRINT*,'Personen Nr.',min_i,min_j,' im Alter von',
     &        Alter(min_i),Alter(min_j)
      END

      PROGRAM Kapitel_5_Aufgabe_4
      IMPLICIT NONE
      INTEGER A(8),B(8),i,j
*     DATA A/2,5,8,4,7,1,6,10/,B/9,8,4,10,3,5,12,11/
* bei Benutzung von DATA entfallen die nächsten 4 Anweisungen
      PRINT*,'bitte 8 Werte für A'
      READ*,(A(i),i=1,8)
      PRINT*,'bitte 8 Werte für B'
      READ*,(B(i),i=1,8)
*
      PRINT*,'In A und B:'
      DO i=1,8
        DO j=1,8
          IF ( A(i) .EQ. B(j) ) PRINT*,A(i)
      ENDDO
      PRINT*,'Weiter mit Enter'
      READ*
      PRINT*,'In B, aber nicht in A:'
      DO i=1,8
        DO j=1,8
* wenn auch nur eines der Aj gleich Bi ist: Schleife vorzeitig verlassen
          IF ( B(i) .EQ. A(j) ) GOTO 10
        ENDDO
* hier wird die j-Schleife beendet, wenn alle Aj ungleich Bi sind
        PRINT*,B(i)
10      CONTINUE
      ENDDO
      PRINT*,'Weiter mit Enter'
      READ*
```

```
      PRINT*,'In A, aber nicht in B:'
      DO i=1,8
        DO j=1,8
* wie oben, nur die Rolle von A und B vertauscht
          IF ( A(i) .EQ. B(j) ) GOTO 20
        ENDDO
* hier wird die j-Schleife beendet, wenn alle Bj ungleich Ai sind
        PRINT*,A(i)
20      CONTINUE
      ENDDO
      PRINT*,'Weiter mit Enter'
      READ*
*
      PRINT*,'Entweder in A oder in B (aber nicht in beiden)'
      PRINT*,'Das sind alle Werte, die in A, aber nicht in B sind'
      PRINT*,'sowie alle Werte, die in B, aber nicht in A sind'
      PRINT*,'also die Vereinigungsmenge der beiden eben gezeigten Werte
     &mengen'
      END
```

```
      PROGRAM Kapitel_5_Aufgabe_5
*    Erweiterung Kap. 4 Aufg. 8:
*    Die n Fibonacci-Quotienten werden in ein Feld  gespeichert
*    nach jedem Durchlauf 1...n werden die Quotienten
*    Fibonacci(i+1) / Fibonacci(i) und   ...i / ...(i-1)
*    in 2 Kolonnen ausgegeben
      IMPLICIT NONE
      INTEGER A,B,C, D,NR, N, I
      REAL Q(100)
      N=20
      D=0
      WHILE(.TRUE.)
        PRINT*,'bitte 1. und 2. Fibonacci-Zahl eingeben; Ende: Strg_z'
        READ*,A,B
*       AT END,QUIT
*       ohne diese Anweisung gibt Strg_z einen  Error - end of file
        D=D+1
        PRINT*,'Durchgang Nr.',d
        NR=2
        WHILE(.TRUE.)
          IF(NR.GT.N) QUIT
          C=A+B
          PRINT*,'Fibonacci-Zahl Nr.',nr,c
          Q(NR)=REAL(C)/REAL(B)
          A=B
          B=C
          NR=NR+1
        ENDWHILE
        PRINT*,'weiter mit Enter'
        READ*
        I=2
        PRINT*,'    Quotienten i / i-1                  i+1 / i'
        WHILE(.TRUE.)
          IF(I.GT.N-1) QUIT
          PRINT*,Q(I),Q(I+1)
          I=I+2
        ENDWHILE
        PRINT*,'Ende eines Durchlaufs von',n,' Fibonacci-Zahlen'
      ENDWHILE
      PRINT*,'Ende des Programms nach',d,' Durchgängen'
      END
```

```
      PROGRAM Kapitel_5_Aufgabe_6
      IMPLICIT NONE
      INTEGER Temperaturen (4,7), Max(4), Min(4),
     &  Max_4_Wochen, Min_4_Wochen
      REAL Durchschnitt(4), Durchschnitt_4_Wochen
      INTEGER i,j
* Eingabe
      DO i=1,4
        PRINT*,'bitte 7 Temperaturen für die',i,'. Woche'
        READ*, ( Temperaturen(i,j), j=1,7)
      ENDDO
* Verarbeitung
      Durchschnitt_4_Wochen = 0
      Max_4_Wochen          = -273
      Min_4_Wochen          = 100
      DO i=1,4
        Durchschnitt(i) = 0
        Max(i)          = -273
        Min(i)          = 100
        DO j=1,7
          Durchschnitt(i) = Durchschnitt(i) + Temperaturen(i,j)
          IF ( Max(i) .LT. Temperaturen(i,j) )
     &         Max(i)  =   Temperaturen(i,j)
          IF ( Min(i) .GT. Temperaturen(i,j) )
     &         Min(i)  =   Temperaturen(i,j)
        ENDDO
        Durchschnitt(i) = Durchschnitt(i) / 7
        Durchschnitt_4_Wochen = Durchschnitt_4_Wochen + Durchschnitt(i)
        IF ( Max_4_Wochen .LT. Max(i) ) Max_4_Wochen = Max(i)
        IF ( Min_4_Wochen .GT. Min(i) ) Min_4_Wochen = Min(i)
      ENDDO
      Durchschnitt_4_Wochen = Durchschnitt_4_Wochen / 4
* Ausgabe
      PRINT*,'                                Durch.      Max         Min'
      DO i=1,4
        PRINT*,'Woche ', I, Durchschnitt(i),Max(i),Min(i)
      ENDDO
      PRINT*,'                           ---------------------------------'
      PRINT*,'4-Wochen-Statistik:',
     &   Durchschnitt_4_Wochen, Max_4_Wochen, Min_4_Wochen
      END

      PROGRAM Kapitel_5_Aufgabe_7
      IMPLICIT NONE
      INTEGER Temperaturen (4,7),Max(4),Min(4),Max_Tag(7),Min_Tag(7),
     &        Max_4_Wochen, Min_4_Wochen
      REAL Durchschnitt(4), Durchschnitt_Tag(7), Durchschnitt_4_Wochen
      INTEGER i,j
* Eingabe
      DO i=1,4
        PRINT*,'bitte 7 Temperaturen für die',i,'. Woche'
        READ*, ( Temperaturen(i,j), j=1,7)
      ENDDO
* Verarbeitung
      Durchschnitt_4_Wochen = 0
      Max_4_Wochen          = -273
      Min_4_Wochen          = 100
```

```
      DO i=1,4
        Durchschnitt(i) = 0
        Max(i) = -273
        Min(i) = 100
        DO j=1,7
          Durchschnitt(i) = Durchschnitt(i) + Temperaturen(i,j)
          IF ( Max(i) .LT. Temperaturen(i,j) )
     &         Max(i)  =   Temperaturen(i,j)
          IF ( Min(i) .GT. Temperaturen(i,j) )
     &         Min(i)  =   Temperaturen(i,j)
        ENDDO
        Durchschnitt(i) = Durchschnitt(i) / 7
        Durchschnitt_4_Wochen = Durchschnitt_4_Wochen + Durchschnitt(i)
      ENDDO
      Durchschnitt_4_Wochen = Durchschnitt_4_Wochen / 4
* Durchschnitt, Max, Min für jeden Tag
      DO j=1,7
        Durchschnitt_Tag(j) = 0
        Max_Tag(j) = -253
        Min_Tag(j) = 100
        DO i=1,4
          Durchschnitt_Tag(j) = Durchschnitt_Tag(j) + Temperaturen(i,j)
          IF ( Max_Tag(j) .LT. Temperaturen(i,j) )
     &         Max_Tag(j)  =   Temperaturen(i,j)
          IF ( Min_Tag(j) .GT. Temperaturen(i,j) )
     &         Min_Tag(j)  =   Temperaturen(i,j)
          IF ( Max_4_Wochen .LT. Max(i) ) Max_4_Wochen = Max(i)
          IF ( Min_4_Wochen .GT. Min(i) ) Min_4_Wochen = Min(i)
        ENDDO
        Durchschnitt_Tag(j) = Durchschnitt_Tag(j) / 4
      ENDDO
* Ausgabe
      PRINT*,'                              Durch.        Max         Min'
      DO i=1,4
        PRINT*,'Woche ', I, Durchschnitt(i),Max(i),Min(i)
      ENDDO
      PRINT*,'                           ---------------------------------'
      PRINT*,'4-Wochen-Statistik',
     &   Durchschnitt_4_Wochen, Max_4_Wochen, Min_4_Wochen
* Ausgabe Tagesstatistik
      PRINT*
      PRINT*,'Tagesstatistik              Durch.        Max         Min'
      DO j=1,7
        PRINT*,'Tag', J, Durchschnitt_Tag(i),Max_Tag(i),Min_Tag(i)
      ENDDO
      PRINT*,'1=Mo, 2=Di, 3=Mi, 4=Do, 5=Fr, 6=Sa, 7=So'
      END
```

```
      PROGRAM Kapitel_6_Aufgabe_1
* benötigt SUBROUTINE Umkehr
      IMPLICIT NONE
      INTEGER A(10)
      DATA A /1,2,3,4,5,6,7,8,9,10/
      PRINT*,'A:'
      PRINT*,A
      CALL Umkehr (A)
      PRINT*,'A umgedreht:'
      PRINT*,A
      END
```

```
      SUBROUTINE Umkehr(V)
      IMPLICIT NONE
      INTEGER V(10),W(10),i
      DO i=1,10
        W(11-i) = V(i)
      ENDDO
      DO i=1,10
        V(i) = W(i)
      ENDDO
      END

      PROGRAM Kapitel_6_Aufgabe_2
* benötigt FUNCTION Det
      IMPLICIT NONE
      REAL Testmatrix(2,2), Det
      DATA Testmatrix/ 1.0, 2.0, 3.0, 4.0 /
      PRINT*,'Die Determinante der Testmatrix ist', Det(Testmatrix)
      END
      REAL FUNCTION Det(X)
      IMPLICIT NONE
      REAL X(2,2)
      Det = X(1,1)*X(2,2) - X(2,1)*X(1,2)
      END

      PROGRAM Kapitel_6_Aufgabe_3
* benötigt FUNCTION Det von Aufg. 2
      IMPLICIT NONE
      REAL A(2,2),Rechte_Seite(2), Loesung(2)
      INTEGER IFLAG
      DATA A/ 2., -4., 3., 8./, Rechte_Seite / 12.,4./
      CALL Cramer(A,Rechte_Seite,Loesung,IFlag)
* Auswertung von IFLAG
      IF ( IFLAG .EQ. 0 ) THEN
          PRINT*,'unendlich viele Lösungen vorhanden'
      ELSEIF ( IFLAG .EQ. 2 ) THEN
          PRINT*,'keine Lösung vorhanden'
      ELSEIF ( IFLAG .EQ. 1 ) THEN
          PRINT*,'Die Lösung ist:', Loesung
      ELSE
          PRINT*,'ungültiger Wert von IFLAG!'
      ENDIF
      END
      SUBROUTINE Cramer(Koeff,RS,Loesung,IFLAG)
      IMPLICIT NONE
      REAL Koeff(2,2),RS(2),Loesung(2),Det_Koeff,Det_RSx,Det_RSy,Det
      INTEGER IFLAG
      Det_Koeff = Det(Koeff)
      Det_RSx   = RS(1)      * Koeff(2,2) - RS(2)      * Koeff(1,2)
      Det_RSy   = Koeff(1,1) * RS(2)      - Koeff(2,1) * RS(1)
      IF ( Det_Koeff .EQ. 0.0 ) THEN
        IF ( Det_RSx .EQ. 0.0 .OR. Det_RSy .EQ. 0.0 ) THEN
*          unendlich viele Lösungen
          IFLAG = 2
        ELSE
*         keine Lösungen
          IFLAG = 0
        ENDIF
      ELSE
        IFLAG = 1
        Loesung(1) = Det_RSx / Det_Koeff
        Loesung(2) = Det_RSy / Det_Koeff
      ENDIF
      END
```

```
      PROGRAM Kapitel_6_Aufgabe_4
* benötigt SUBROUTINE RANDI
      IMPLICIT NONE
      INTEGER Zufallsfeld(60),i,j
      Zufallsfeld(1) = 7**5
      CALL RANDI(Zufallsfeld,0,65535)
      DO i=0,59,6
        PRINT*,(Zufallsfeld(i+j),j=1,6)
      ENDDO
      END
*
      SUBROUTINE RANDI(R,j,k)
      IMPLICIT NONE
      INTEGER R(60),j,k,n
      DO n=2,60
        R(n) = MOD ( 25173 * R(n-1),65536)
      ENDDO
      IF ( .NOT. (j.EQ.0 .AND. k.EQ.65536 ) ) THEN
* nur falls nicht 0...65536 angegeben war: Zahlen umrechnen
        DO n=1,60
          R(n) = R(n) / 65535.0 * ( k-j+1 ) + j
        ENDDO
      ENDIF
      END

      PROGRAM Kapitel_6_Aufgabe_5
* benötigt SUBROUTINE RANDI von Aufg. 4
      IMPLICIT NONE
      INTEGER Zufall(0:239),i,j,k,n,Wurf(6,6),Summe(2:12)
      DATA Wurf/36*0/,Summe/11*0/
*             4 mal je 60 Zufallszahlen ...
      DO n=0,3
*             1. Element wird "zufällig" auf 7**n gesetzt
        Zufall(60*n) = MOD( 7 ** n, 65536 )
        CALL RANDI(Zufall(60*n),1,6)
      ENDDO
* 100 Würfe mit je 2 Würfeln: je 2 aufeinanderfolgende Zufallszahlen
      DO i=0,199,2
* war der Wurf z.B. (6,6), so ergibt die folgende Anweisung:
*       Wurf(     6   ,     6     ) = Wurf(     6   ,     6    ) + 1
        Wurf(Zufall(i),Zufall(i+1)) = Wurf(Zufall(i),Zufall(i+1)) + 1
      ENDDO
      PRINT*,'Tabelle der Wurfhäufigkeiten (i,j) = Element (i,j)'
      DO i=1,6
        PRINT*, (Wurf(i,j),j=1,6)
      ENDDO
* Häufigkeiten der Würfe mit der Summe i+j = Element (i+j) von Summe'
      DO i=1,6
        DO j=1,6
          Summe (i+j) = Summe(i+j) + Wurf(i,j)
      ENDDO
      DO k=2,12
        PRINT*,'Rel.Häufigkeit der Würfe mit Summe',k,' =',Summe(k)/100.
      ENDDO
      END

      PROGRAM Kapitel_6_Aufgabe_6
* benötigt SUBROUTINE Insert
      IMPLICIT NONE
      INTEGER Feld(20),I, Ins
      DATA Feld/20*999999/
*               -999999 wenn absteigend sortiert werden soll
      PRINT*,'Es sollen 20 Zahlen einsortiert werden'
```

```
      DO i=1,20
      PRINT*,'bitte Zahl',i,'(max.6 Ziffern) zum Einsortieren angeben'
        READ, Ins
        CALL Insert(Feld,Ins)
      ENDDO
      PRINT*,'Die folgenden Werte müssen aufsteigend sein:'
      PRINT*,Feld
      END

      SUBROUTINE Insert(K,Ins)
      IMPLICIT NONE
      INTEGER K(20),Ins,n,i
* vergleiche, bis Ins nicht mehr kleiner als die bereits sortierten Werte
      n = 1
      WHILE ( Ins .GT. K(n) .AND. n .LT. 20)
*                .LT.    würde absteigend sortieren
        n = n + 1
      ENDWHILE
* Platz schaffen für das neue Element
      DO i=20,n+1,-1
        K(i) = K(i-1)
      ENDDO
* da kommt es hin
      k(n) = Ins
      END
```

```
      PROGRAM Kapitel_6_Aufgabe_7
* benötigt SUBROUTINEn : EINGABE,BERECHNE,AUSGABE; FUNCTIONs : UEBER,FAK
* Aufgabenstellung: Pascalsches Dreieck          1
*                                              1   1
*                                            1   2   1
*                                          1   3   3   1
*                                        .   .   .   .   .
*   Jede Zahl ergibt sich als Summe der Zahlen schräg rechts und links
*                                              ( n )   ( n )      ( n )
*   Zeile n enthält die Binomialkoeffizienten  ( 0 ) , ( 1 ) ,.., ( n )
*   Definition der Binomialkoeffizienten ( a )      a !
*                          a ueber b:    ( b ) =  --------------
*                                                  b ! * ( a-b) !
*   Die Summe in jeder Zeile ist 2**n    (1.Zeile soll n=0 haben)
*   maximal 9. Ordnung;     es wird nur ganzzahlig gerechnet
*   die Ausgabe erscheint in der Form  .
*                                      .  .
*                                      .  .  .   usw.
*   da die symmetrische Dreiecksgestalt schwierig zu programmieren ist
*   (Fortgeschrittene können es mit Hilfe von Formaten versuchen)
      IMPLICIT NONE
      INTEGER Anzahl, Zeile(11), Ordnung, Max_Ordnung
      PARAMETER ( Max_Ordnung = 10 )
      LOGICAL Ende /.FALSE./
      Anzahl = 0
* Endlosschleife für mehrfache Eingabe
      WHILE(.TRUE.)
        CALL EINGABE(Ordnung,Max_Ordnung,Ende)
        IF ( Ende ) QUIT
* die folgende Anweisung verläßt die Schleife (weiter hinter ENDWHILE)
        Anzahl = Anzahl +1
        CALL Berechne( Zeile,Ordnung)
      ENDWHILE
      PRINT*
      PRINT*,'es wurden',Anzahl,' Pascalsche Dreiecke erzeugt'
      PRINT*,'Ende des Programms Pascaldreieck'
      END
```

```
      SUBROUTINE EINGABE (Ordnung, Max_Ordnung, Ende)
      IMPLICIT NONE
      LOGICAL Ende
      INTEGER Ordnung, Max_Ordnung
      PRINT*
      PRINT*,'bitte Ordnung des Pascalschen Dreiecks angeben'
      READ*, Ordnung
      IF(Ordnung.GT.Max_Ordnung) THEN
        PRINT*,'maximale Ordnung ist',Max_Ordnung
        Ordnung = Max_Ordnung
      ELSEIF(Ordnung .LT. 0) THEN
        Ende = .TRUE.
        RETURN
      ENDIF
      PRINT*,'Pascalsches Dreieck der Ordnung',Ordnung
      END

      SUBROUTINE BERECHNE (Zeile,Ordnung)
      IMPLICIT NONE
      INTEGER Zeile(11),Ordnung,UEBER,n,m
      PRINT*,'           1'
      DO n=1,Ordnung
        DO m=0,n
* n ueber m: also  ( 1 )  ( 1 ) bis ( 1 ) ... (ordnung) ... (ordnung)
*                  ( 0 )  ( 1 )     ( 0 )     (    0  )     (ordnung)
          Zeile(m+1)= UEBER(n,m)
        ENDDO
        CALL AUSGABE (Zeile,n+1)
      ENDDO
      END

      SUBROUTINE AUSGABE (Zeile,x)
      IMPLICIT NONE
      INTEGER Zeile(11),j,x
*        WRITE(6,'(10I8)') ( Zeile(J),J=1,x)  besseres Druckbild!
        PRINT*, ( Zeile(J),J=1,x)
      END

      INTEGER FUNCTION UEBER (a,b)
* berechnet a "über" b, dh.  a! / ( b! (a-b)! )
      IMPLICIT NONE
      INTEGER a,b, FAK
        UEBER = FAK(a) / ( FAK(b) * FAK(a-b) )
      END
      INTEGER FUNCTION FAK(n)
* berechnet n! (n Fakultät)
      IMPLICIT NONE
      INTEGER n, m
      IF(n.LT.0) THEN
        PRINT*,'***ERROR*** N Fakultät für n=',n
        STOP
      ELSE
        FAK=1
        DO m=2,n
          FAK = FAK * m
        ENDDO
      ENDIF
      END
      PROGRAM Kapitel_6_Aufgabe_8
* benötigt die SUBROUTINE Baue_Matrix
      IMPLICIT NONE
      INTEGER Max_n,Max_m,Max_nm,n,m,i,j
      PARAMETER ( Max_n = 6 , Max_m = 6, Max_nm = Max_n * Max_m )
```

```
* Eingabe
      INTEGER Matrix(Max_n,Max_m),Vektor(Max_nm)
      PRINT*,'bitte n und m eingeben'
      READ*,n,m
      IF ( n .GT. Max_n ) THEN
        PRINT*,'n ist maximal', Max_n
        n = Max_n
      ENDIF
      IF ( m .GT. Max_m ) THEN
        PRINT*,'m ist maximal', Max_m
        m = Max_m
      ENDIF
      PRINT*,'bitte', n * m, ' Werte eingeben'
      READ*, ( Vektor(i), i = 1, n*m )
* Verarbeitung
      CALL Baue_Matrix ( Max_n, Max_n, n, m, Vektor, Matrix )
* Ausgabe
      PRINT*,'Ausgabe der aufgebauten Matrix:'
      DO i=1,n
        PRINT*, ( Matrix ( i,j ), j=1,m )
      ENDDO
      END
*
      SUBROUTINE Baue_Matrix ( d1, d2, a,b, X, Y )
* Kapitel_6_Aufgabe_8
      IMPLICIT NONE
      INTEGER d1,d2, a,b, X(a*b), Y(d1,d2), n,i,j
      n = 0
      DO i=1,a
        DO j=1,b
          n = n + 1
          Y (i,j) = x(n)
      ENDDO
      END

      PROGRAM Kapitel_6_Aufgabe_9
* benötigt die SUBROUTINE Merge
      IMPLICIT NONE
      INTEGER Max_m, Max_2m, m1,m2,i
      PARAMETER ( Max_m = 10, Max_2m = 2 * Max_m )
* Eingabe
      INTEGER Menge1 (Max_m), Menge2 (Max_m), M12(Max_2m)
      PRINT*,'bitte m1 und m2 eingeben'
      READ*,m1,m2
      IF ( m1 .GT. Max_m ) THEN
        PRINT*,'m1 ist maximal', Max_m
        m1 = Max_m
      ENDIF
      IF ( m2 .GT. Max_m ) THEN
        PRINT*,'m2 ist maximal', Max_m
        m2 = Max_m
      ENDIF
      PRINT*,'bitte', m1, ' Werte eingeben'
      READ*, ( Menge1(i), i = 1, m1 )
      PRINT*,'bitte', m2, ' Werte eingeben'
      READ*, ( Menge2(i), i = 1, m2 )
* Verarbeitung
      CALL Merge ( m1,m2, Menge1,Menge2, M12 )
* Ausgabe
      PRINT*,'Ausgabe der gemergten Menge:'
      PRINT*, ( M12 ( i ), i=1,m1+m2 )
      END
```

```
      SUBROUTINE Merge ( a,b, X,Y,Z )
      IMPLICIT NONE
      INTEGER a,b, X(a), Y(b), Z(a+b), i,j,k,n
      i = 1
      j = 1
      k = 1
      WHILE ( i .LE. a .AND. j .LE. b )
        IF ( X(i) .LE. Y(j) ) THEN
          Z(k) = X(i)
          i = i + 1
        ELSE
          Z(k) = Y(j)
          j = j + 1
        ENDIF
        k = k + 1
      ENDWHILE
      IF ( i .GT. a ) THEN
        DO n=j,b
          Z(k) = Y(n)
          k = k + 1
        ENDDO
      ELSEIF ( j .GT. b ) THEN
        DO n=i,a
          Z(k) = X(n)
          k = k + 1
        ENDDO
      ENDIF
      END
```

```
      PROGRAM Kapitel_7_Aufgabe_1
      IMPLICIT NONE
      REAL Temperatur (4,7),Durchschnitt(4),Max(4),Min(4),
     &  Durchschnitt_4_Wochen, Max_4_Wochen, Min_4_Wochen
      INTEGER i,j
* Eingabe
      DO i=1,4
        PRINT*,'bitte 7 Temperatur für die',i,'. Woche'
        READ*, ( Temperatur(i,j), j=1,7)
      ENDDO
* Verarbeitung
      Durchschnitt_4_Wochen = 0.0
      Max_4_Wochen          = 0.0
      Min_4_Wochen          = 0.0
      DO i=1,4
        Durchschnitt(i) = 0.0
        Max(i)          = -273.0
        Min(i)          = 100.0
        DO j=1,7
          Durchschnitt(i) = Durchschnitt(i) + Temperatur(i,j)
          IF ( Max(i) .LT. Temperatur(i,j) )
     &         Max(i)  =   Temperatur(i,j)
          IF ( Min(i) .GT. Temperatur(i,j) )
     &         Min(i)  =   Temperatur(i,j)
        ENDDO
        Durchschnitt(i) = Durchschnitt(i) / 7.0
        Durchschnitt_4_Wochen = Durchschnitt_4_Wochen + Durchschnitt(i)
        Max_4_Wochen          = Max_4_Wochen          + Max(i)
        Min_4_Wochen          = Min_4_Wochen          + Min(i)
      ENDDO
      Durchschnitt_4_Wochen = Durchschnitt_4_Wochen / 4.0
      Max_4_Wochen          = Max_4_Wochen          / 4.0
      Min_4_Wochen          = Min_4_Wochen          / 4.0
* Ausgabe
```

```
      PRINT*,'           Mo    Di    Mi    Do    Fr    Sa    So
     &Durch. Max   Min'
      DO i=1,4
        WRITE(6,'(A,I2,7F6.1,7X,3F6.1)') 'Woche', I,
     &  (Temperatur(i,j),j=1,7),Durchschnitt(i),Max(i),Min(i)
      ENDDO
      WRITE(6,'(58X,16(''-'')/ A, 38X ,3F6.1)') '4-Wochen-Statistik',
     &   Durchschnitt_4_Wochen, Max_4_Wochen, Min_4_Wochen
      END

      PROGRAM Kapitel_7_Aufgabe_2
      IMPLICIT NONE
      REAL Temperatur (4,7),Durchschnitt(4),Max(4),Min(4),
     &  Durchschnitt_4_Wochen, Max_4_Wochen, Min_4_Wochen
      REAL Durchschnitt_Tag(7),Max_Tag(7),Min_Tag(7)
      INTEGER i,j
* Eingabe
      DO i=1,4
        PRINT*,'bitte 7 Temperatur für die',i,'. Woche'
        READ*, ( Temperatur(i,j), j=1,7)
      ENDDO
* Verarbeitung
      Durchschnitt_4_Wochen = 0.0
      Max_4_Wochen          = 0.0
      Min_4_Wochen          = 0.0
      DO i=1,4
        Durchschnitt(i) = 0.0
        Max(i) = -273.0
        Min(i) = 100.0
        DO j=1,7
          Durchschnitt(i) = Durchschnitt(i) + Temperatur(i,j)
          IF ( Max(i) .LT. Temperatur(i,j) )
     &         Max(i)  =   Temperatur(i,j)
          IF ( Min(i) .GT. Temperatur(i,j) )
     &         Min(i)  =   Temperatur(i,j)
        ENDDO
        Durchschnitt(i) = Durchschnitt(i) / 7.0
        Durchschnitt_4_Wochen = Durchschnitt_4_Wochen + Durchschnitt(i)
        Max_4_Wochen          = Max_4_Wochen          + Max(i)
        Min_4_Wochen          = Min_4_Wochen          + Min(i)
      ENDDO
      Durchschnitt_4_Wochen = Durchschnitt_4_Wochen / 4.0
      Max_4_Wochen          = Max_4_Wochen          / 4.0
      Min_4_Wochen          = Min_4_Wochen          / 4.0
* Durchschnitt, Max, Min für jeden Tag
      DO j=1,7
        Durchschnitt_Tag(j) = 0.0
        Max_Tag(j) = -253
        Min_Tag(j) = 100
        DO i=1,4
          Durchschnitt_Tag(j) = Durchschnitt_Tag(j) + Temperatur(i,j)
          IF ( Max_Tag(j).LT.Temperatur(i,j) )Max_Tag(j) = Temperatur(i,j)
          IF ( Min_Tag(j).GT.Temperatur(i,j) )Min_Tag(j) = Temperatur(i,j)
          ENDDO
        Durchschnitt_Tag(j) = Durchschnitt_Tag(j) / 7.0
        ENDDO
* Ausgabe
      PRINT*,'           Mo    Di    Mi    Do    Fr    Sa    So
     &Durch. Max   Min'
      DO i=1,4
        WRITE(6,'(A,I2,7F6.1,7X,3F6.1)') 'Woche', I,
     &  (Temperatur(i,j),j=1,7),Durchschnitt(i),Max(i),Min(i)
      ENDDO
```

```
      WRITE(6,'(58X,16(''-'')/ A, 38X ,3F6.1)') '4-Wochen-Statistik',
     &   Durchschnitt_4_Wochen, Max_4_Wochen, Min_4_Wochen
* Ausgabe Tagesstatistik
      PRINT*
      PRINT*,'Tagesstatistik    Durch.  Max   Min'
      DO j=1,7
        WRITE(6,'(A,I2, 12X, 3F6.1)') 'Tag', J,
     &  Durchschnitt_Tag(j),Max_Tag(j),Min_Tag(j)
      ENDDO
      PRINT*,'1=Mo, 2=Di, 3=Mi, 4=Do, 5=Fr, 6=Sa, 7=So'
      END

      PROGRAM Kapitel_7_Aufgabe_3
* benötigt die SUBROUTINE Merge von Kap. 6 Aufg. 9
      IMPLICIT NONE
      INTEGER Max_m, Max_km, Max_k, i,j,k,ii,l
      PARAMETER ( Max_m = 10, Max_k = 5, Max_km = Max_k * Max_m )
* Eingabe
      INTEGER Menge(Max_m,Max_k), m(Max_k),
     &        Menge_gemerged_1(Max_km), Menge_gemerged_2(Max_km)
      PRINT*,'bitte Zahl der zu mergenden Mengen angeben ( >=2 ):'
      READ*,k
      IF ( k .GT. Max_k ) THEN
        PRINT*,'k ist maximal', Max_k
        k = Max_k
      ELSEIF ( k .LT. 2 ) THEN
        PRINT*,'k ist mindestens 2'
        k = 2
      ENDIF
      DO i=1,k
        WRITE(6,'(A,I1,A)') 'bitte m',i,' angeben'
        READ*,m(i)
        IF ( m(i) .GT. Max_m ) THEN
          WRITE(6,'(A,I1,A,I2)') 'm',i,' ist maximal', Max_m
          m(i) = Max_m
        ENDIF
        WRITE(6,'(A,I2,A)') 'bitte',m(i),' aufsteigend sortierte Werte
     &eingeben'
        READ*, ( Menge(j,i), j = 1, m(i) )
* Verarbeitung
        IF ( i .EQ. 2 ) THEN
* Merge die ersten beiden Mengen
          ii = m(1) + m(2)
          CALL Merge ( m(1),m(2),
     &               Menge(1,1), Menge(1,2), Menge_gemerged_1 )
        ELSEIF ( i .GT. 2 ) THEN
* Merge eine weitere ( Menge(1,i) zu der bereits vorhandenen dazu
          CALL Merge ( ii, m(i),
     &               Menge_gemerged_1, Menge(1,i), Menge_gemerged_2)
* die Länge der neuen Menge ist gleich der alten plus m(i)
          ii = ii + m(i)
* die neu entstandene Menge wird an den Platz der alten abgespeichert
          DO l=1,ii
            Menge_gemerged_1(l) =  Menge_gemerged_2 (l)
          ENDDO
        ENDIF
      ENDDO
* Ausgabe
      PRINT*,'Ausgabe der',k,' gemergten Mengen mit',ii,' Elementen:'
      PRINT*, ( Menge_gemerged_1 ( i ), i=1,ii)
      END
      SUBROUTINE Merge ( a,b, X,Y,Z )
* s. oben Kapitel_6_Aufgabe_9
```

Index

Die Zahlen geben die Seiten im Skript an, wo das entsprechende Stichwort nachzulesen ist.

A-Code ... 73
ACCESS= ... 24
ADVANCE= ... 74
Addition ... 11 20 27 81
Algorithmus ... 12 16 17 19 28 35 38 40 49 51 52 53 55
Anfangswert ... 34 36 39 41 43 46 85 87 88
Anweisung, ausführbare ... 87 88
Anweisungs-Teil ... 18
Anweisungsfeld ... 10
Anweisungsnummer ... 10 62
Argument ... 25 50 51 52 53 57 61 88
Argumenten-Liste ... 50 61 88
Arithmetik ... 11 20 21 26
Arithmetik, gemischte ... 21
ASA-Drucksteuerzeichen ... 70
ASCII-Code ... 79
Aufruf, Wirkungsweise ... 61
Aufruf einer Subroutine ... 50 51 88
Ausdruck, arithmetischer ... 88
Ausdruck, logischer ... 28 29 31 32 34 87 88
Ausführen des Programms ... 05 07
Ausführungszeit ... 59
Ausgabe von Integer-Zahlen ... 68 72
Ausgabe von Real-Zahlen ... 71
Ausgabe-Element ... 67 76
Ausgabe-Gerät ... 67
Ausgabe, formatfreie ... 67 68 87
Ausgabe, formatierte ... 67 68 75
Ausgabeposition ... 69
Ausgabeposition, implizite Angabe ... 70
Ausgang aus der Schleife ... 48
Auslöschung ... 27
Ausnahmebedingung ... 38

Bedingung ... 13 28 30 32 34 35 37 40 48 87 88
Bibliothekskonzept ... 57
blank ... 10 69 71 72 73 75 76 77
Block-IF ... 28 29 31
BN-Code ... 73 75
BZ-Code ... 73 75

CALL-Anweisung ... 51
CASE-Anweisung ... 57
Character-Ausdruck ... 78 88
Character-Operator ... 78 88
Character-String ... 10 25
Character-Variable ... 76 78
Character-Werte ... 78
compile time ... 12 17 18
compile time error ... 12 14
Compiler ... 12 14 15 17 18 21 22 34 36 42 57 69 73 90
CONTINUE-Anweisung ... 64 65
Cursor ... 09 14 73 75 86
D-Code ... 71 72
DATA-Anweisung ... 46 73
Datenende ... 64
Debugger ... 18 90

Deklaration ... 13 18 21 22 46 50 57 59 61 85 87 88
Deklaration, explizite ... 22
Deklaration von Konstanten ... 57
Deklarations-Liste ... 87 88
Deklarations-Teil ... 18
Dezimalpunkt ... 13 14 17 20 63 67 72 73 88
Dezimalsystem ... 26
Dezimalzahl ... 14 17 26 40
Dimensionierung ... 41 45 46 52 59 60 61 65
Dimensionierung zur Ausführungszeit ... 59
Dimensions-Liste ... 87 88
Division ... 11 12 15 20 21 25 27 28 33 36 38 40
Division durch 0 ... 12 15 28 36
Division, ganzzahlige ... 20
Divisionsrest ... 20 25
DO-Anweisung ... 35 36
DO-Block ... 36
DO-Schleife ...
  35 36 37 39 40 41 43 45 46 60 61 64 67 75 81 88
DO-Schleife, implizite ... 43 64 67
DO-Schleifen-Steuergröße ... 61
DO-Parameter ... 36
DO-Variable ... 36 65

E-Code ... 71 72
E/A-Anweisung ... 73 82 88
E/A-Befehl ... 75 76
E/A-Befehl, neuer ... 75
E/A-Einheit ... 88
E/A-Gerät ... 63 90
E/A-Liste ... 63 64 73 75 87 88
EBCDI-Code ... 79
Editieren ... 07 15 24
Editor ... 07 08 09 90
Ein- und Ausgabe ... 07 08 09 11 24 43 63 67 76 90
Eingabe-Datenelement ... 63 64 76
Eingabe-Gerät ... 63
Eingabe-Liste ... 23 64 76
Eingabe, formatfreie ... 63 64 66 87
Eingabe, formatierte ... 63 66
Einlesen einer unbekannten Anzahl von Daten ... 64
END= ... 56 57 64 65
END-Anweisung ... 50 51
End-Of-File ... 64
Endebedingung ... 48 87
Endebedingungen, mehrere ... 37
Endlosschleife ... 08 34 39 57
Endwert ... 36 43 46 88
Entscheidung ... 11 28
ERR= ... 64
error ... 08 12 17 18 64 74
Erste-Buchstaben-Regel ... 22 60
executable statement ... 18
execution time ... 12 17 18
execution time error ... 15 18
EXIT-Anweisung ... 48
Exponent ... 19 20 21 25 63 67 71 72 73 88
Exponentenbereich ... 21
Exponentialfunktion ... 25
Exponentiation ... 20 88
extension ... 09 18
EXTERNAL-Deklaration ... 60 61

Feld, Ein- und Ausgabe ... 43
F-Code ... 71 72
Fehlermeldung ... 12 14 15 18 42 65
Fehlerquellen ... 26
Feld, eindimensionales ... 41
Feld, gerufenes ... 61
Feld, höher dimensioniertes ... 45
Feld, übergebenes ... 61
Feld, 2-dimensional ... 43
Feldelement ... 41 42 44 46 61 62 67 79 83 88
Feldname ... 42 45 46 88
Feldvariable ... 62 67
File ... 07 08 09 24
File, Ausdrucken ... 09
File, internes ... 82
Filehandling ... 09
Filename ... 70
Filename, Erweiterung ... 07
FMT= ... 64 66 68 70 71 72
FORMAT-Anweisung ... 63 66 67 68 69 88
Formatcode ... 69 72 74 75 76 77 83 86 87 88
Formatcode, autonomer ... 75
Formatcode, beschreibender ... 75
Format-Codes ... 85
Formate, Regeln ... 75
formatfrei ... 63 64 66 67 68 87
formatiert ... 07 63 66 67 68 73 75 83
Formatinfo ... 63 64 66 68
Formatnummer ... 10 62
Formatsteuerung ... 66
Fortran-Anweisung ... 10 14 15 63 67
Fortran-Namen ... 18 22
Fortran-Standard ... 09 22
Fortran-Standard-Erweiterung ... 81
Fortran-Typ ... 17 87
Fortran-Programm ... 09 10 12 18 50 51 62
Fortran90-Standard ... 09 18 22 24 34 36 40 48 57 74 81 89
Fortsetzungsspalte ... 10
FUNCTION-Anweisung ... 50 87
Function-Name ... 62
Function-Unterprogramm ... 54
Funktion, eingebaute ... 25 79 80
Funktion, numerische ... 25 88
Funktion, trigonometrische ... 25
Funktionen ...
  18 20 25 32 50 54 55 57 60 61 62 78 79 81 84 88
Funktionsnamen ... 50 54 57 61 87
Funktionsnamen-Liste ... 87
Funktionsaufruf ... 54
Funktionswert ... 47 54 61 62 79 83 86

G-Code ... 72 73
Genauigkeit ... 11 22 26 27 71
Geräte-Nr. ... 24 67 82 83 87
gerundet ... 77
global ... 62
GOTO-Anweisung ... 28
Hauptprogramm ... 50 51 53 58 59 60 62 75 87
Hauptprogrammstruktur ... 58 87

I-Code ... 68 69 72 73 79
IBM-Compiler ... 36
IBM-Großrechner ... 70 79
IEEE-Standard ... 21
IF-Anweisung ... 26 28
IF-Block ... 28 29 30 31
Index ... 25 41 42 43 44 45 46 47 79 80 81 85 86
Indexgrenze, untere ... 45
Integer-Ausdruck ...20 42
Integer-Konstante ... 22 88
Integer-Variable ... 22
Integer-Zahl ... 20 21 22 68 69 70 72
IMPLICIT NONE ... 22
iterative Berechnung ... 16

Klammerebene ... 74
Klammer ... 20 32 34 40 41 50 74 88
Kommentar ... 10 12 15 17 18 49 57 81 82 87
Konkatenierung ... 78 79 88

L-Code ... 73
Leerelement ... 64
Leerzeichen ...
  07 10 17 23 43 49 64 68 69 70 78 79 80 81 82 88
Listing ... 08 09 12 15 24 90
Listing-File ... 08 09 15 24
Logarithmus ... 25 27
Logical-Konstante ... 88
lokal ... 18 57 62 85

Maschinensprache ... 12 17
Max-Dimension ... 88
Meldungen ... 08 09 12 18
Modul ... 57
MS/DOS ... 07
Multiplikation ... 11 20 27 37 40

Name, Gültigkeitsbereich ... 57 62
Name des Programms ... 12
Namen der eingebauten Funktionen ... 62
Namen, globale ... 62
Namen, lokale ... 62
Netzwerk-Drucker ... 09
nicht (log. Operator) ... 32
Normal-Record ... 76
Nullelement ... 64
Nummernfeld ... 10

oder (log. Operator) ... 32
oder, exklusives ... 30
OPEN-Anweisung ... 24 70
Operation, arithmetische ... 11 32
Operator, arithmetischer ... 88
Operator, logischer ... 28 32 88

P-Code ... 73
PARAMETER-Anweisung ... 57 65
Parameter ... 50 51 53 54 57 58
Parameter-Liste ... 50 57 60 61 87 88
Polymorphismus ... 25
PRINT-Anweisung ... 12 16 19 67
Priorität ... 20 21 28 32

privat ... 57
Problem ... 11 12 17 38 41 46 48 58 82
Programm, rufendes ... 51 57 76
Programmbibliothek ... 57
PROGRAM-Anweisung ... 51
Programm-Kopf ... 12
Programmablauf ... 28
Programmlisting ... 05 07 08
Programmname ... 07 08 87 90

Quellprogramm ... 08 12
QUIT-Anweisung ... 40 48

READ-Anweisung ... 21 23 63 64 66 75
Real-Ausdruck ... 42 78
Real-Division ... 21
Real-Grundrechenarten ... 21
Real-Konstante ... 22 71 88
Real-Variable ... 41
Real-Zahl ... 15 20 21 26 71 72
Real-Zahl, ganzzahliger Teil ... 21
Real-Zahl, gebrochener Teil ... 21
Record ... 73 75 82
Record, neuer ... 73 75 83 88
Recordlänge ... 76
RETURN-Anweisung ... 51

S-Code ... 73 75
Schachtelung ... 30
Schleife, regulärer Ausgang ... 48
Schleife, zweiter Ausgang ... 48
Schleifensteuerung ... 34 35 38 48
Schnelldrucker ... 11 70
Schnittstelle ... 57
Schrittweite ... 36 43 84 87 88
SELECT-Anweisung ... 57
Skalenfaktor ... 73
Sortieren ... 79
SP-Code ... 73 75
SS-Code ... 73 75
Standard-Ausgabe ... 67 68 76
Standard-Eingabe ... 63 66
Standard-Einheit ... 63 64 68
Standard-Funktions-Name ... 62
Steueranweisung ... 28
Steuerung ... 28 34 35 38 48 66 67 73
STOP-Anweisung ... 36 51
String ... 10 21 68 73 76 78 79
String-Konstante ... 75 88
Stringverarbeitung ... 25 78 79 81
strukturiert ... 48
SUBROUTINE-Anweisung ... 50 87
Subroutine-Name ... 62
Subroutine-Unterprogramm ... 50
Substring ... 78 79 88
Subtraktion ... 11 20 27
Syntax-Fehler ... 14 17 18

T-Code ... 69 73 75
Tabulator ... 09 73 75 88
Text ... 10 18 67
trace ... 90

traceback ... 15 90
Typ CHARACTER ... 18 66 73 78 79 82 87
Typ COMPLEX ... 17 25 87
Typ INTEGER ... 17 36 42 67 73 87
Typ LOGICAL ... 18 48 73 87
Typ REAL ... 13 17 46 87
Typ REAL*8 ... 22 87
Typen, Voreinstellung ... 22
Typ-Deklaration ... 21 46
Typ-Umwandlungen ... 25

Umlenken der Standard-Ein/Ausgabe ... 24
und (log. Operator) ... 32
undefinierter Wert, Ausgabe ... 15
undefinierter Wert, Rechnen ... 15
Unit ... 24 64 68 79 82 84 86 87 90
UNIT= ... 24 64 68 79 82 84 86 87
Unterprogramm, externes ... 60 61
Unterprogramm, öffentlicher Teil ... 57
Unterprogramm, privater Teil ... 57
Unterprogramm-Namen ... 62
Unterprogrammeinheit ... 57
Unterprogrammstruktur ... 87
UProg-Liste ... 61

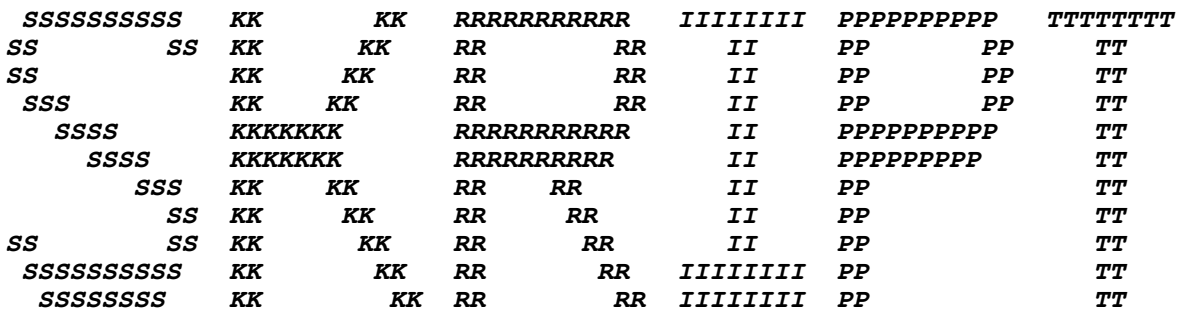
Variable, einfache ... 62
Variablen-Liste ... 64 66 68
Vektor ... 41 46 57 58 59 62
Vergleichs-Operator ... 14 28 32 88
von:bis ... 57 78 88 90

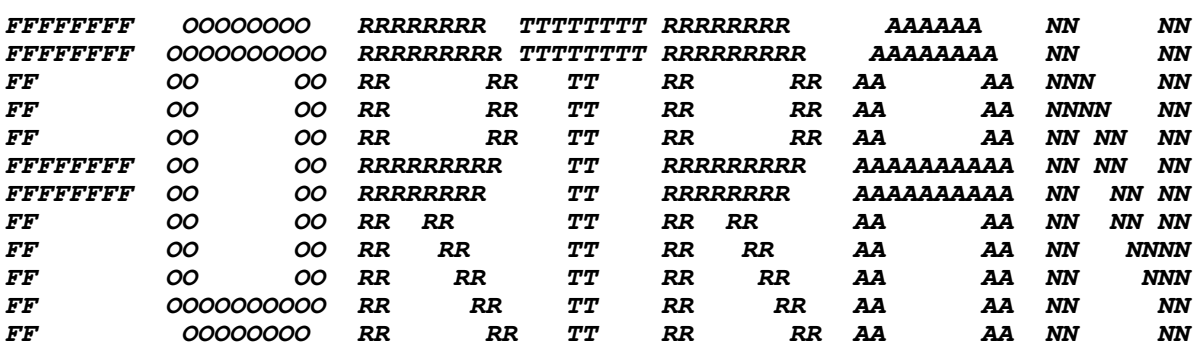
warning ... 18
WATFOR ... 07 08 09 12 18 22 24 90
WEDIT ... 07
Werte, Übergabe ... 61
Werte-Liste ... 87 88
WHILE-Anweisung ... 13 18 34
WHILE-Schleife ... 34 35 87 88
Wiederholungsfaktor ... 46 64 72 74 75 88
Workspace ... 07 08 09
Workstation ... 21 26 79
WRITE-Anweisung ... 67 68 75 82

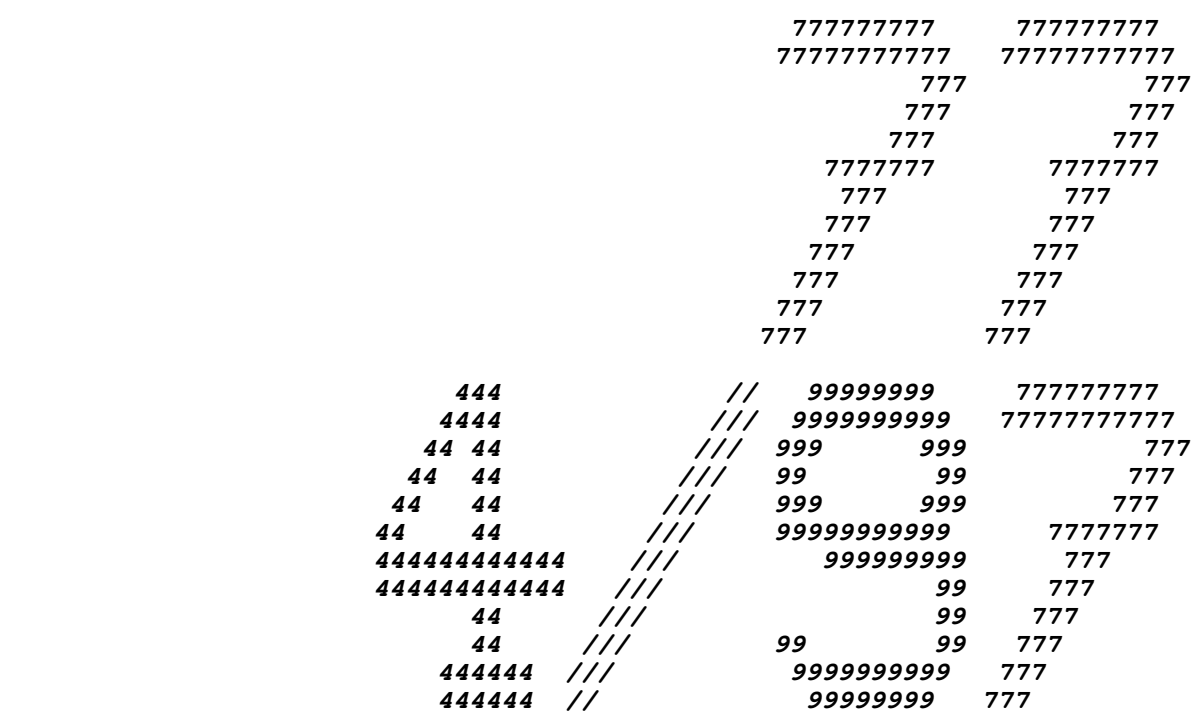
X-Code ... 69 73 75

Z-Code ... 73
Zahl, ganze ... 17 20 26
Zahl, komplexe ... 17 25 72
Zahl mit Dezimalpunkt ... 13 20
Zahlenbereich ... 20 68 71
Zeichenkette ... 78
Zuweisung von Werten ... 22

: ... 73 75
/ ... 73 75
$ ... 74 75







© S.Vetter URZ Heidelberg 1986,1994,1997

Verantwortlich für den Inhalt: Siegfried Vetter
Datum der letzten Änderung: 25.03.1997