Jülich Supercomputing Centre

Programmieren in Fortran 90/95

Übungsaufgaben

Gerd Groten

Benutzerhandbuch · FZJ-JSC-BHB-0133

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
FORSCHUNGSZENTRUM

FORSCHUNGSZENTRUM JÜLICH GmbH
Jülich Supercomputing Centre
D-52425 Jülich, Tel. (02461) 61-6402

Benutzerhandbuch

Programmieren in Fortran 90/95
Übungsaufgaben

Gerd Groten

FZJ-JSC-BHB-0133

3. Auflage

(letzte Änderung: 27.05.1999)

Publikationen des JSC stehen in druckbaren Formaten (PDF auf dem WWW-Server des Forschungszentrums unter der URL: <http://www.fz-juelich.de/jsc/files/docs/> zur Verfügung. Eine Übersicht über alle Publikationen des JSC erhalten Sie unter der URL: **<http://www.fz-juelich.de/jsc/docs>** .

Beratung

Tel: +49 2461 61 -nnnn

Auskunft, Nutzer-Management (Dispatch)

Das Dispatch befindet sich am Haupteingang des JSC, Gebäude 16.4, und ist telefonisch erreichbar von

Montag bis Donnerstag	8.00 - 17.00 Uhr
Freitag	8.00 - 16.00 Uhr

Tel. 5642 oder 6400, Fax 2810, E-Mail: **dispatch.jsc@fz-juelich.de**

Supercomputer-Beratung

Tel. 2828, E-Mail: **sc@fz-juelich.de**

Netzwerk-Beratung, IT-Sicherheit

Tel. 6440, E-Mail: **junet-helpdesk@fz-juelich.de**

Rufbereitschaft

Außerhalb der Arbeitszeiten (montags bis donnerstags: 17.00 - 24.00 Uhr, freitags: 16.00 - 24.00 Uhr, samstags: 8.00 - 17.00 Uhr) können Sie dringende Probleme der Rufbereitschaft melden:

Rufbereitschaft Rechnerbetrieb:	Tel. 6400
Rufbereitschaft Netzwerke:	Tel. 6440

An Sonn- und Feiertagen gibt es keine Rufbereitschaft.

Fachberater

Tel. +49 2461 61 -nnnn

Fachgebiet	**Berater**	**Telefon**	***E-Mail***
Auskunft, Nutzer-Management, Dispatch	E. Bielitza	5642	***dispatch.jsc@fz-juelich.de***
Supercomputer	W. Frings	2828	***sc@fz-juelich.de***
JuNet/Internet, Netzwerke, IT-Sicherheit	T. Schmühl	6440	***junet-helpdesk@fz-juelich.de***
Web-Server	Dr. S. Höfler-Thierfeldt	6765	***webmaster@fz-juelich.de***
Backup, Archivierung	U. Schmidt	1817	***backup.jsc@fz-juelich.de***
Fortran	D. Koschmieder	3439	***fortran.jsc@fz-juelich.de***
Mathematische Methoden	Dr. B. Steffen	6431	***mathe-admin@fz-juelich.de***
Statistik	Dr. W. Meyer	6414	***mathe-admin@fz-juelich.de***
Mathematische Software	Dr. B. Körfgen, R. Zimmermann	6761, 4136	***mathe-admin@fz-juelich.de***
Graphik	Ma. Busch, M. Boltes	4100, 6557	***graphik.jsc@fz-juelich.de***
Videokonferenzen	M. Sczimarowsky, R. Grallert	6411, 6421	***vc.jsc@fz-juelich.de***
Oracle-Datenbank	M. Wegmann, J. Kreutz	1463, 1464	***oracle.jsc@fz-juelich.de***

Die jeweils aktuelle Version dieser Tabelle finden Sie unter der URL:
<http://www.fz-juelich.de/jsc/allgemeines/beratung>

Inhaltsverzeichnis

Kapitel 1

Schreibweise von Quellprogrammen

1.1 include

1.1.1 Wie genau soll's sein?

Gegeben sei folgendes Programm:

```
  implicit real (a-h,o-z)
  a = -1
  b = acos(a)
  call sub_program(b)
end
!------------------------------
subroutine sub_program(x)
  implicit real (a-h,o-z)
  write(unit=*,fmt=*) x
end subroutine sub_program
```

Ersetzen Sie die `implicit`-Anweisungen durch ein `include`. In der eingezogenen Datei soll z.B. stehen können:

```
implicit doubleprecision (a-h,o-z)
```

Probieren Sie die Wirkung aus!

1.2 grammatische Grundelemente

1.2.1 Namen

Welche der folgenden Namen sind legale Fortran-90-Namen?

```
Wort        Word93
REST        128
NGC132i5    Niemals-Nie
HaltEin!    F_D_P_
Gleitkommakonstantenexponent
```

1.3 spaltengerechtes Quellprogramm

1.3.1 Nicht normal

Was entspricht in dem folgenden spaltengerechten Programm nicht der Norm?

```
*---------------------------------
      c hara cter*10 name
      N A M E = 'Ganten' //
     @          'bein'
      Zahl = 17 . 0
      if ( (ZAHL/3)*3 == zahl )goto1
      write(unit=*,fmt=*) N A M E, Zahl
      call FUSSB
     call SPIEL ! Was wird geCALLt ?
    1 End!e des Liedes
     ! --- Programmende ---
```

1.3.2 "Happy" old Fortran

Was bedeutet das folgende in Fixed-Form-Fortran?

```
      integer :: i = 8
      do 25 i = 1.25
        write(unit=*,fmt=*) i
   25 continue
```

1.3.3 tückisches altes Fortran

Wie werden die folgenden Anweisungen bei der spaltengerechten und bei der freien Quellprogrammform aufgenommen?

```
      logical :: l
      integer :: i=0,j=7
      read *,  l
      if (l) then i = 2
      else j = i endif
      write(unit=*,fmt=*) i,j
      end
```

Würde `implicit none` etwas ändern? Kann man das Programm korrigieren, ohne die Anzahl der Zeilen zu ändern?

1.3.4 Spitzfindig

Wie werden die folgenden Anweisungen bei der spaltengerechten und bei der freien Quellprogrammform aufgenommen?

```
*23456789
      logic all
      logi call ogical
      l= .true.; ! aha ;logical= .false.
```

Kapitel 2

Die Datentypen `logical` und `integer`

2.1 Der Datentyp `logical`

2.1.1 Änderung der Logik

Es gibt eine bekannte Umformung von „nicht (a oder b)“. Beweisen Sie per Programm, dass diese Umformung auch in Fortran stimmt!

Untersuchen Sie ebenso Umformungen von „nicht (a und b)“ , „nicht (a oder nicht-b)“ , „ nicht (a und nicht-b)“!

Da DO-Schleifen und der Integer-Datentyp noch nicht besprochen wurden, folgt hier ein Muster zur Erzeugung der Wahr-Falsch-Kombinationen:

```
integer :: i, j
logical :: a, b, c
do i=0,1
  a = i == 1
  do j=0,1
    b = j == 1
    c = ...
    write(unit=*,fmt=*) a,b,c
  end do
end do
```

2.2 Klassen vordefinierter Funktionen

2.2.1 Eine Frage der Logik

Nur eine Frage: Was bewirkt „merge(.false._byte, .true._byte, logische_Variable)“ ? Kann man das auch mit einer logischen Operation und der Funktion `logical` erreichen?

2.3 Integer-Variabeln und -Konstantennamen

2.3.1 Probe aufs Exempel

Versuchen Sie, trotz `implicit none` einer nicht-deklarierten Variable k einen Wert zuzuweisen oder auch eine andere Variable l einfach auszugeben, ohne Zuweisung! Was geschieht ohne `implicit none`?

Versuchen Sie, in Fortran90/95 einem Konstantennamen mit `data` den Wert z’ffffffff’ zu geben! Geht das? Versuchen Sie, einem Konstantennamen nachträglich durch Zuweisung einen Wert zu geben!

Wie reagieren die Compiler?

2.4 Das Integer-Zahlenmodell, Abfragefunktionen

2.4.1 Kalkuliertes Risiko

Lassen Sie vorsichtig die Anzahl der Möglichkeiten, einen Lottoschein 6 aus 49 auszufüllen, (49*48*47*46*45*44) / (1*2*3*4*5*6) , berechnen!

Benutzen Sie `integer`-Arithmetik; ändern Sie - wenn erlaubt - vielleicht die Reihenfolge der Rechnung, berechnen Sie aber nichts vorher selbst!
Versuchen Sie auch, dem Problem durch eine höhere Integer-Genauigkeit gerecht zu werden!

2.4.2 Katastrophe

Was geschieht bei Ihrem Rechner, wenn Sie schreiben:

```
integer :: i, j, k
i = 32768
j = 65536
k = i*j
write(unit=*,fmt=*) k
```

Versuchen Sie, zunächst durch Überlegung zu formulieren, was zu erwarten oder zu befürchten ist.

2.4.3 Wie weit kann man gehen?

Deklarieren Sie zehn Variablen mit Hilfe von `selected_int_kind`(r), r=2,3,4,5,9,10,13,14,18,19. Geben Sie von diesen Variabeln `kind, radix, digits, huge, range` aus! Versuchen Sie, vorher durch Überlegung herauszufinden, welche Werte ausgegeben werden! Bei einer der Abfragefunktionen hilft Überlegung allerdings nichts. IBM RS/6000 (xlf95, nagf95) und CRAY (f90).

2.4.4 Eine Unzahl

Die Ergebnisse sind bei dieser Aufgabe nicht von der Norm definiert.

Geben Sie bitte einem normalen Integer-Konstantennamen den Wert -2147483648. Geht das beim Compiler xlf95? Auch bei nagf95? Wie ist die Reaktion, wenn Sie -2147483647-1 schreiben?

Was geschieht bei diesen beiden Compilern, wenn im Programm durch einen Integer-Konstantennamen mit dem Wert 0 dividiert wird?

```
integer,parameter :: i = 0
integer,save      :: j = 0
write(unit=*,fmt=*) 1/i   ! Ist das moeglich?
write(unit=*,fmt=*) 1/j   ! Wird hier etwas ausgegeben?
```

2.4.5 Gesetzeslücke

Der f90-Compiler von Cray hat folgende Anweisungen ohne weiteres nach Wunsch übersetzt. Wie reagiert der xlf95- und nagf95-Compiler?

```
integer,parameter:: n = 4
integer,parameter:: m = selected_int_kind(n)
integer(kind=m),parameter :: i = huge(i)
write(unit=*,fmt=*) i
```

Suchen Sie eine alternative, weniger zweifelhafte Schreibweise!
Wie reagieren die Compiler, wenn Sie dann n=18 setzen?

2.5 Interpretation und Auswertung von Integer-Ausdrücken

2.5.1 Schaden durch Nebenwirkungen

Das folgende Programm mit einer selbstgeschriebenen externen Funktion ist nach dem, was gerade über die Auswertungsreihenfolge gesagt wurde, falsch. Warum? Kommt bei den Compilern xlf95 und nagf95 dasselbe Ergebnis?

```
! Nicht ueberall brauchbar, nicht F
program rando
  implicit none
  integer :: int_random, iseed
  iseed = 4711
  write(unit=*,fmt=*) iseed + int_random(iseed)
end program rando

function int_random(is) result(erg)
  integer :: is ! Mit Absicht ohne intent
  integer :: erg
  is = ibclr( is*84331461 + 1815266769, bit_size(is)-1)
  erg = is
end function int_random
```

Man kann aus dem Programm einen brauchbaren Pseudo-Zufallszahlen-Generator machen; aber nicht mit dieser Funktion, die ihren Parameter ändert.

2.6 Zuweisung

2.6.1 Vom Falschen des Richtigen

Vergleichen Sie per Programm durch eine Vergleichsoperation == oder /=:

Bei i=121 und j=22: i*(1+2+1) mit j*j !

Bei i=12321 und j=333: i*(1+2+3+2+1) mit j*j !

Bei i=1234321 und j=4444: i*(1+2+3+4+3+2+1) mit j*j !

Und noch einmal weiter in dem Schema!

Was geschieht, wenn die Teile i und j nicht erst auf Variablen gespeichert werden, sondern unmittelbar im Ausdruck als Konstanten stehen, beim IBM-Compiler xlf95, beim NAGWare-Compiler nagf95?

2.6.2 Mathematik für Hasen

Wann ist Ostern? „J“ sei das Jahr post Christi natu.

$$
\begin{aligned}
a &= J \ mod \ 19 \\
b &= J \ mod \ 4 \\
c &= J \ mod \ \ 7 \\
\\
m &= \left[\frac{8\left[\frac{J}{100}\right]+13}{25} \right] - 2 \\
\\
s &= \left[\frac{J}{100}\right] - \left[\frac{J}{400}\right] - 2 \\
\\
M &= (15 + s - m) \ mod \ 30 \\
N &= (6 + s) \ mod \ 7 \\
d &= (M + 19a) \ mod \ 30 \\
D &= \begin{cases} 28 & falls \ d = 29 \\ 27 & falls \ d = 28 \ und \ a \geq 11 \\ d & sonst \end{cases} \\
e &= (2b + 4c + 6D + N) \ mod \ 7
\end{aligned}
$$

Ostern fällt (nach Gauß) im Jahr J auf den (D+e+1)-ten Tag nach dem 21. März. Drucken Sie bitte das Datum aus! Das Jahr kann eingelesen werden: read (unit=*,fmt=*) J.

2.7 Listengesteuerte Ein/Ausgabe

2.7.1 Verstanden?

Welchen Wert haben die Variablen am Ende?

```
integer :: i,j,k,l,m,n
real    :: g

read(unit=*,fmt=*) i,j
read(unit=*,fmt=*) k
read(unit=*,fmt=*) l,m,n
read(unit=*,fmt=*)
read(unit=*,fmt=*) g
```

Zugehörige Datenzeilen:

```
1      2        3          4
    5       6        7
8      9
10     11
12

13.8
   14.8   15.76   16.
```

2.8 Ein/Ausgabe: Formate für Integer und Logical

2.8.1 Wie es Euch gefällt

Schreiben Sie je eine Fortran90-Anweisung zur Ausgabe eines Integer-Wertes vom Typparameter `selected_int_kind`(2 und 4 und 9 und 18).
Nebeneinander soll jeweil im i-, b-, o-, z-Format ausgegeben werden. Es geht nicht in F. Die Zeilen sollen vier Kolonnen bilden. Die nichtdezimalen Formate sollen führende Nullen haben.
Als Beispiel sollte die größte Zahl der jeweiligen Darstellung genommen werden, einmal positiv und einmal mit negativem Vorzeichen.

Kapitel 3

Kontroll-Anweisungsgruppen

3.1 `if`-Fallunterscheidung (wenige Fälle)

3.1.1 Gewichtsabweichnung

Wir wollen annehmen, dass das spezifische Gewicht eines Menschen nahezu 1 kg/dm**3 ist. Das Volumen in cm**3 ist dann so gross wie das Gewicht G in Gramm. Wie Physiker es tun, abstrahieren wir die Gestalt eines Menschen. Wir nehmen an, dass seine Breite 2/9 mal seiner Größe L ist, die wir in cm angeben. Seine Dicke ist dann (9*G)/(2*L**2).

Schreiben Sie ein Programm, welches das Gewicht und die Groesse einliest und die Dicke berechnet. Wenn Sie 10 cm ist, soll der Mensch als normalgewichtig gelten. Ist sie 9 bzw. 11, so soll das als erträgliche Abweichung gelten.

Lösen Sie das Problem mit `if` und `case`! Welches dieser Konstrukte ist bei diesem Problem angemessen?

3.2 `case`-Fallunterscheidung (u.U. viele Fälle)

3.2.1 Zufall oder Tendenz?

Es sollen recht viele nichtnegative Zahlen aus nur einer Ziffer eingelesen werden. Wenn sie mehr Ziffern haben, soll nur die letzte genommen werden. Mit unseren jetzigen Mitteln, also ohne Felder, soll festgestellt werden, wie oft am Ende der Eingabedatei jede Ziffer vorkam.

Aufgabe „Rational denken“, später.

3.3 `do`-Wiederholung

3.3.1 Zwei bemerkenswerte Zahlen

Berechnen Sie mit Fortran:

$$4^4 + 3^3 + 8^8 + 5^5 + 7^7 + 9^9 + 0^0 + 8^8 + 8^8$$

Setzen Sie $0^0 = 0$! Was fällt am Ergebnis auf? Versuchen Sie herauszubekommen, ob es auch eine vierstellige Zahl mit derselben Eigentümlichkeit gibt!

3.3.2 Mathematik an Kaiser Karls Pfalz

Eine Aufgabe von Alcuin aus York, Bischof von Tours, Förderer der karolingischen Renaissance:

Es sind 100 Scheffel Weizen so zwischen 100 Leuten aufzuteilen, dass jeder Mann 3, jede Frau 2 und jedes Kind 1/2 Scheffel erhält. Wie viele Männer, Frauen und Kinder sind es?

Es gibt 7 Lösungen. Alcuin gab aber nur eine an (11,15,74).

3.3.3 Leonardo Pisano (Fibonacci) 1180- 1240

Jemand hat 30 Vögel für 30 Münzen gekauft. Für drei Spatzen zahlte er 1 Münze, für zwei Wildtauben ebenfalls 1 Münze, und für jede Zuchttaube 2 Münzen. Wieviele Vögel jeder Art hat er gekauft?

3.3.4 Obststapeln

Jemand hat Gummikugeln quadratisch auf den Boden gelegt, in einer Anordnung $n \cdot n$. Weil er keinen Platz mehr hat, will er sie als Tetraeder stapeln mit m Kugeln an jeder Kante; das gibt $m \cdot (m+1) \cdot (m+2)/6$ Kugeln im Tetraeder.

Für welche(s) n kann er ein vollständig aufgebautes Tetraeteder bilden? Wieviele Kugeln sind beteiligt?

Ein triviales Beispiel ergibt sich mit $n=2$ und $m=2$. Gibt es noch ein geeignetes Paar (n,m)?

Lassen Sie die Schleifen zur Erzeugung von m und n von 3 an laufen bis zu einer willkürlichen oberen Grenze. Wenn diese Grenze ohne Erfolg erreicht wird, soll eine Meldung ausgedruckt werden. Wenn eine Lösung gefunden wurde, soll das Programm diese ausgeben und dann aufhören.

3.3.5 Auf der Demo

Stellen wir uns in Zehnerreihen auf, ist einer zu wenig. Stellen wir uns in Neunerreihen auf, ist ebenfalls einer zu wenig. So geht es weiter bis zu Zweierreihen, wo auch einer fehlt. Wieviele sind wir? (Unter 5000).

3.3.6 Ein Kinderlied

Welche Werte nehmen im folgenden Programmstück IZYK und KK an? Versucht, es an Hand eines Beispiels zu verstehen! Könnte statt `modulo` genau so gut `mod` benutzt sein?

```
integer           :: kk, izyk, I
integer,parameter ::izyanf=2, izyend=4,&
             klipp=47, klapp=11, iend=60

izyk= Zahl zwischen izyanf und izyend &
      einschl. (z.B. izyend)

kk = klipp oder klapp (z.B. klipp)

do i = 1,iend
   kk =  klipp + klapp - kk
   izyk = izyanf+modulo(izyk-izyanf+1, &
     izyend-izyanf+1)
   - - - - -
end do
```

3.3.7 Maximum-Minimum-Thermometer

Von der Standard-Eingabedatei soll von jeder Zeile die erste Zahl listengesteuert gelesen werden, bis zum Ende der Datei. Es soll sich um ganze Zahlen handeln. Dabei soll die kleinste und die größte der Zahlen bestimmt werden und die ganze Zahl, welche dem arithmetischen Mittel aller eingelesenen Zahlen am nächsten kommt.

3.3.8 Nochmals Fibonacci, Leonardo Pisano

Die ersten beiden Glieder der Fibonacci-Folge sind 0 und 1. Jedes weitere Glied ist als Summe der beiden vorhergehenden definiert. Schreiben Sie ein Programm, das die Glieder der Folge zusammen mit ihrer Position innerhalb der Folge ausgibt, und zwar solange wie es bei dem gewählten Integer-Typ geht.

3.3.9 Non_plus_ultra

Drucken Sie nebeneinander n und n! aus. Gerechnet werden soll mit einer einstellbaren Integer-Genauigkeit, soweit es bei dieser Genauigkeit geht. Die Schleife mit n soll maximal bis 30 laufen.

3.3.10 Wer hat die meisten Schäfchen?

Drucken Sie untereinander alle Teiler der Zahl 9240 aus!

3.3.11 300 vor Christus

Programmieren Sie den Euklidischen Algorithmus zur Berechnung des größten gemeinsamen Teilers ggT zweier ganzer Zahlen m und n!

Man bestimmt den Divisionsrest r von m/n. Wenn der Rest nicht 0 ist, tut man so, als wollte man den ggT von n und r berechnen.

Wenn der Rest 0 ist, dann ist der letzte Divisor der ggT von m und n.

3.4 Implizites do bei der Ein/Ausgabe

3.4.1 Deutsche Industrienorm A4

Schreiben Sie zwei Seiten gut lesbar voll mit dem Wertepaar { i ,modulo(i,10) * modulo(i+1,10) * modulo(i+2,10) / 6 } !
i wird dabei von 1 an hochgezählt.

Eine Zeile soll 80 Zeichen fassen, eine Seite 62 Zeilen. Ein Wertepaar soll nicht getrennt werden, es sollen aber mehrere Wertepaare in jeder Zeile stehen.

Aufgabe „Morse-Alphabet“, später.

Kapitel 4

Bitweise Verarbeitung von Integer-Größen

4.1 Vordefinierte Bit-Manipulations-Funktionen

4.1.1 Metamorphose

Versuchen Sie durch Tests herauszubekommen, was die folgenden drei Anweisungen bewirken!

```
i= ieor(i,j); j= ieor(i,j); i= ieor(i,j)
```

4.1.2 Elementare Computerlogik

I sei eine Integer-Variable. Kann man bei der RS/6000 *-i* ohne das Minuszeichen bilden? Versuchen Sie's!

4.1.3 Der digitale Euklid

Schreiben Sie ein Programmstück zur Bestimmung des größten gemeinsamen Teilers von I,J (beide positiv)!

Binärer GGT-Algorithmus:

Zunächst werden I,J beide sooft (K-mal) durch zwei geteilt (um 1 Bit nach recht geschoben), bis wenigstens eins von beiden ungerade wurde. Ist von I,J eines noch gerade, so wird es ebenfalls immer wieder durch 2 geteilt, bis es ungerade geworden ist. Nun wird L=| I-J| gebildet.

Solange L von 0 verschieden ist, werden nun folgende Schritte wiederholt: L (das zunächst immer gerade ist) wird sooft durch 2 geteilt, bis es nicht mehr durch 2 teilbar ist. Das Größere von I und J wird durch dieses L ersetzt; es wird dann wieder L=| I-J| genommen.

Wenn endlich L zu 0 wurde, ist 2**K*I der größte gemeinsame Teiler der ursprünglichen I,J.

Dieser Algorithmus ist vielleicht geeignet für solche Rechner, die sich mit dem Teilen von Integerzahlen bei voller Länge schwertun (z.B. CRAY, `selected_int_kind`(18) bei RS/6000).

Kapitel 5

`character`, Zeichenfolgen

5.1 Vordefinierte Funktionen zur Zeichenverarbeitung

5.1.1 Ein paar Handvoll Zeichen

Geben Sie alle Zeichen aus, vom 0-ten bis 255-ten. Fällt Ihnen etwas auf?

5.1.2 Däumling

Eine (unter Umständen sehr lange) Zeichenfolge bestehe nur aus Buchstaben. Ihr Programm soll das erste Vorkommen des alphabetisch kleinsten Buchstabens finden. Position und Zeichen!

5.1.3 Im Zentrum

Eine Zeichenfolgenvariable soll so geschoben werden, dass am Anfang und Ende gleich viele Leerzeichen stehen, oder wenigstens beinahe gleich viele.

5.1.4 Im Briefkopf

Wandeln Sie das Datum, welches von DATE_AND_TIME geliefert wird, in unsere deutsche Schreibweise um. Drucken Sie den neuen String aus! Erklärung der vordefinierten Subroutine ganz hinten in den Kursusunterlagen.

5.1.5 Wortklauberei

Eine Zeichenfolge *zeile* bestehe aus mehreren Worten. Worte enthalten keine Kommas und keine Leerzeichen. Drucken Sie die Zeichenfolge und die einzelnen Worte aus! Drucken Sie neben dem Wort aus, ob es eine Integer-Zahl beliebiger Länge sein kann.
Man kann *zeile* wie folgt einlesen:

```
character(len=80) :: zeile
integer           :: ios
read(unit=*,fmt="(a)",iostat=ios) zeile
```

5.1.6 Aneinanderklammern

Suchen Sie in einer Zeichenfolge die äußersten Klammern, seien es nun () oder $\{\}$ oder[] oder $<>$! Drucken Sie den Inhalt aus! Wenn es gar kein Klammernpaar gibt, oder zur ersten öffnenden Klammer keine schließende da ist, soll das Programm das mitteilen.

5.2 Lesen aus Zeichenfolgen, Schreiben auf Zeichenfolgen

5.2.1 Zur Freude der Numerologen

Diese Aufgabe ist als Beispiel in den Kursfolien verwendet.
Gibt es ein pythagoräisches Dreieck (das ist eines mit rechtem Winkel und ganzahligen Seiten), dessen Fläche, dezimal geschrieben, aus lauter gleichen Ziffern besteht?

```
a = m**2-n**2
b = 2*m*n
c = m**2+n**2
```

Man bekommt daraus alle pythagoräischen Dreiecke (Seiten a, b, c) .

Lassen Sie bei ersten Tests des Programms *n* und *m* nicht größer als 50 werden!

5.2.2 Mastermind

Schreiben Sie ein Programm, das als „Schiedsrichter“ beim Mastermind-Spiel fungiert! Spielregeln:

- Der Rechner gibt eine positive Zahl mit 4 Ziffern vor. Er entnimmt die Zahl `system_clock`. Wenn *count* weniger als 4 Stellen hat, nimmt er *count_max* - *count*, jeweils die letzten 4 Stellen. Wenn man das Spiel sehr gut beherrscht, kann man die 4 zu einer 5 machen.
- Die mitspielende Person versucht, diese Zahl zu erraten, indem sie in 10 Versuchen vierziffrige Zahlen eingibt.
- Nach jedem Versuch meldet das Programm, 1) wieviele Ziffern zwar in der zu erratenden Zahl vorkomen aber noch nicht an der richtigen Stelle stehen, 2) wieviele schon an der richtigen Stelle stehen.

5.3 Listengesteuerte Ein/Ausgabe von Zeichenfolgen

5.3.1 Morse-Alphabet

Drucken Sie eine Zeile voll mit `'.---.---.'` mit wenigstens einem Leerzeichen zwischen den einzelnen SOS-Funkzeichen. Benutzen Sie das implizite DO!

Siehe auch ”rational denken” bei Funktionen!

5.4 Character: Ein/Ausgabe mit Formaten

5.4.1 Zählen, was Zahl ist

Die Eingabe bestehe aus Zeilen mit maximal 80 Zeichen. Ihr Programm soll in diesen Zeilen alle Worte als ganze Zahlen zu lesen versuchen. Worte sind durch Komma oder Leerzeichen voneinander gegrennt. Die Zahlenwerte aller als Zahl interpretierbaren Worte sollen für jede Zeile addiert und die Summe ausgegeben werden.

Dabei soll *vier* oder *IV* nicht als Zahl gelten, wohl aber z.B. *4* und *4.4E4* .

5.4.2 Alle in einen Topf

Lesen Sie mehrere Zeichenfolgen mit Längen bis 99, die jeweils in einer Zeile stehen und so { ... } begrenzt sind! Wenn sonst noch etwas in der Zeile steht, braucht es nicht beachtet zu werden. Die Begrenzungszeichen dürfen im Inneren nicht vorkommen.

Die Zeichenfolgen im Innern der Begrenzungszeichen sollen in einer Zeichenfolge TOPF von 4096 Zeichen hintereinander abgespeichert werden. Vor jeder Zeichenfolge steht als Dezimalzahl in Zeichenform die Länge der Zeichenfolge. Zum Überprüfen sollen alle Zeichenfolgen aus TOPF ausgegeben werden, wenn nichts mehr hinzukommt oder TOPF voll ist.

Irgendwie muß man an dem provisorischen Längenfeld der Zeichenfolge, die zum Schluß als nächste gekommen wäre, das Ende festzustellen sein.

Kapitel 6

Die Datentypen `real` und `complex`

6.1 Reelle Zahlen: Zahlenmodell

6.1.1 Unschärfe

Bilden Sie *n*0.1* mit *n* von 5 bis 95 (auch 4 bis 94) in Schritten von 10! Dabei soll zum einen das Produkt, so wie es da steht, berechnet und ausgegeben werden, zum anderen als Summe *s* von *n* Summanden. Drittens *n*/10.0 . Was fällt auf?

Die Summe s soll mit „real :: s“ deklariert sein.

Benutzen Sie den xlf95-Compiler einmal ohne Optionen (oder auch debF), ein andermal mit -qfloat=hssngl -O3 -qstrict ! Vergleichen Sie weiter die Ergebnisse beim nagf95-Compiler !

Zu dem hier bemerkten Verhalten sei auch auf das Kapitel über IEEE verwiesen.

6.2 Normale und doppeltgenaue Gleitkommazahlen, der Typparameter

6.2.1 Who is who

Deklarieren Sie eine reelle Variable mit 6, eine zweite mit 7, eine dritte mit 8 Dezimalstellen Genauigkeit (CRAY: 13, 14, 15) ! Welche von Ihnen haben den gleichen Typparameter? Welcher entspricht dabei `real`, welcher `double precision`?

Welchen Typparameter bekommt man mit der Angabe `selected_real_kind(6,50)` ?

6.3 Vordefinierte Abfragefunktionen zum Zahlenmodell

6.3.1 Angst vor Nähe

Schreiben Sie ein Programm, das einen Wert x einliest und daraus den Wert x/(1+x) berechnet und ausgibt. Wenn x sehr nahe an -1 liegt, soll das Programm eine Fehlernachricht ausgeben und versuchen, einen neuen Wert für x einzulesen.

6.3.2 echte Typen

Geben Sie für i von 5 bis 33 `selected_real_kind(i)` aus! Welche Typparameter (also wirklich vorhandene reelle Typen) gibt es beim xlf95-Compiler, beim nagf95-Compiler und bei CRAY?

Lassen Sie in einem zweiten Programm zu jedem der echten Typen bei jedem der Compiler folgende Bestimmungsstücke ausgeben: `digits, huge, epsilon, maxexponent, minexponent, precision, radix, range, tiny` ! Sind die Typen bei CRAY alle *echt* oder in Wirklichkeit nur formal vorhanden?

Siehe auch: **Eingeweideschau** in dem Kapitel über IEEE.

6.4 Vordefinierte Funktionen zur Gleitkomma-Manipulation

6.4.1 Offenes Intervall

Ein offenes Intervall (z.B.] -1, +1 [ ) soll in 100 durchnumerierte gleichgroße Teilintervalle zerlegt werden. Wird nun eine Zahl *x* aus dem Intervall eingegeben, z.B. `nearest`(1.0_rp, -1.0_rp) bei 64-Bit-IEEE-Arithmetik, dann soll das Programm berechnen, in welchem Teilintervall dieses *x* liegt.

6.4.2 Das optimale π

Berechnen Sie `acos`(-1.0) , die nächst kleinere Realzahl und die nächst größere Realzahl! Mit allen drei Zahlen soll `sin`(...) berechnet werden.

Dasselbe auch bei doppelter Genauigkeit.

Am einfachsten ist es, wenn Sie einen Typparameter benutzen und zweimal übersetzen.

6.4.3 Leitersprossen

Zählen Sie alle Gleitkommazahlen zwischen 3 und 4 und die zwischen 1023 und 1024!

6.4.4 Lemminge

Mit folgender Formel kann man das Verhalten von Tierpopulationen simulieren:

```
Folgepopulation = Fp (zwischen 0 und 100)
Ausgangspopulation = Ap (zwischen 0 und 100)
Lebenswert = Lw (zwischen 0 und 2)

Fp = Lw*Ap*(100 - Ap)/50
```

Starten Sie mit Ap = 90! Tabellieren Sie maximal 60 Iterationen. Wenn zwei aufeinder folgende Populationen sich um weniger als 1000 Einheiten der letzten Stelle unterscheiden, dann hören Sie schon vorher auf.

Das Bevölkerungsverhalten ist sehr verschieden, je nachdem welchen Lebenswert man wählt: <0.5, zwischen 0.5 und 1.5, zwischen 1.5 und 1.9142, >1.9142.

6.5 Vordefinierte numerisch / mathematische Funktionen

6.5.1 Blick aufs Meer

Wie weit kann man von einem Turm, Berg oder Ballon (Höhe : h Meter) aufs Meer oder eine Ebene hinaus sehen? Gemeint ist die Entfernung auf der gekrümmten Erdoberfläche. Die Erde soll als Kugel betrachtet werden mit einem Radius von 6367467 Metern.

Drucken Sie die Ergebnisse tabellarisch aus für Höhen von 1 m, 2 m, 5 m, 10 m, 20 m, 50 m ... 50000 m.

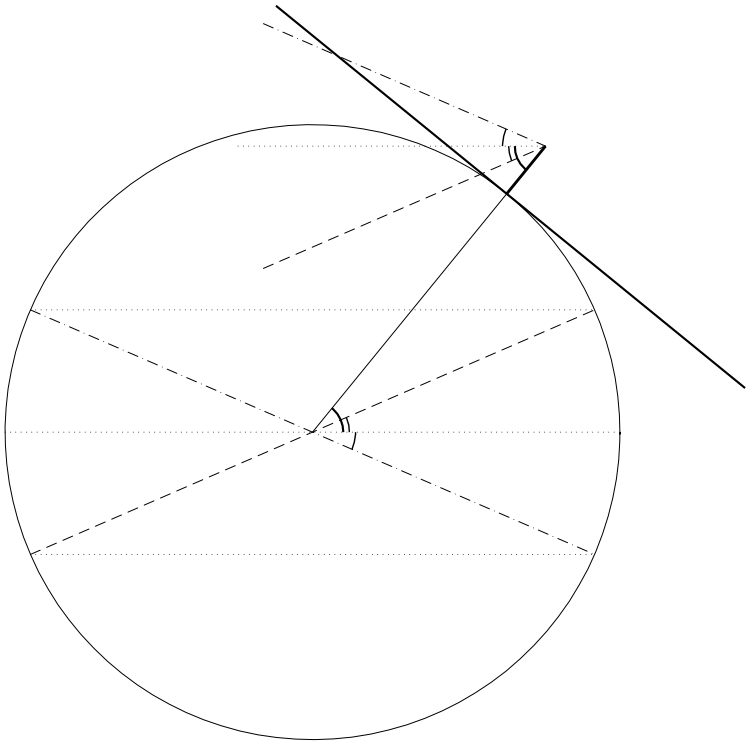
Machen Sie eine Kontrollrechnung für eine Höhe von *Radius*($\sqrt{2}$-1)*. Das müsste ein halber Erdquadrant sein. Das Meter ist ursprünglich (zur Zeit Napoleons) so festgelegt worden, dass ein Erdquadrant 10000 km misst.

6.5.2 Aufgaben der Vorsokratiker

Der Mittagschatten eines 1 Meter langen lotrechten Stabes ist irgendwo zur Wintersonnenwende 3.5937 Meter lang, zur Sommersonnenwende aber nur 0.5217 Meter.

Wie gross ist die geographische Breite dieses Ortes? Wie lang ist der Schatten des Stabes zur Zeit der Tag- und Nachtgleiche? Bei welchem Grad liegen die Wendekreise?

Die Aufgabe stammt vielleicht schon von Anaximander (geb. 610 vor Christus in Milet).



6.5.3 Ein altägyptischer Brunnen

Das folgende Problem geht auf eine Aufgabe zurück, die man 1912 bei Ausgrabungen im Nildelta an den Wänden eines Tempels der Pharaonenzeit fand. Wer Priester des Gottes Ra werden wollte, wurde in eine echte Klausur eingeschlossen, aus der er nur wieder entlassen wurde, wenn er die Aufgabe löste, oder wenn er vor Hunger zusammenbrach.
In einem runden Brunnen mit dem Durchmesser D (Ellen) steht ein langer Schilfstengel der Länge A (Ellen) unten gegen die Wand des Zylinders und oben an die Gegenseite gelehnt.
Genau gegenüber ist ein anderer, kürzerer Schilfstengel der Länge B (Ellen) in gleicher Art hingestellt.
In dem Brunnen steht das Wasser C Ellen hoch. Die beiden Schilfstengel kreuzen sich gerade in der Höhe des Wasserspiegels.
Mit diesen Informationen soll man den Durchmesser D berechnen, wie hoch Schilfstengel B am Brunnenrand reicht (X), und wie hoch Schilfstengel A reicht (Y).
Die Prüflinge hatten damals den Brunnen und die Schilfstengel wirklich vor sich und eine Elle zum Messen, weil sie nicht einmal den Satz des Pythagoras kannten. Wir dagegen können nichts nachmessen, haben aber den Lehrsatz von Pythagoras und die Strahlensätze als Hilfsmittel.

Ansatz:

$$X^2 + D^2 = B^2$$
$$Y^2 + D^2 = A^2$$
$$\frac{X}{C} = \frac{D}{D_1}$$
$$\frac{Y}{C} = \frac{D}{D_2}$$
$$mit$$
$$D_1 + D_2 = D$$

Dieser Ansatz führt zu folgender Gleichung für X :

$X^4+P_3X^3+P_2X^2+P_1X+P_0 = 0$

oder in der Klammerung nach Horner, die für numerische Rechnungen besser ist:

$(((X + P_3)X + P_2)X + P_1)X + P_0 = 0$

Die Koeffizienten P sind dabei wie folgt aus den Problemparametern bestimmt:

$P_3 = -2*C$
$P_2 = A^2-B^2$
$P_1 = P_3*P_2$
$P_0 = C^2*P_2$

Aus geometrischen Überlegungen folgt, dass die gesuchte Lösung für X zwischen C und 2*C liegt. Suchen Sie durch wiederholte Halbierung (binary search) des Intervalls die Lösung X zu finden! Hinweis: f(links)*f(mitte) >0, <0, == 0 ? Berechnen Sie aus X auch D und Y!

Zur Kontrolle seien zwei Beispiele mit Lösung angegeben:

```
A=11.9, B=7.0, C=3.0|X=4.2, D=5.6, Y=10.5
A=10.5, B=8.7, C=3.5|X=6.0, D=6.3, Y=8.4
```

6.5.4 Vieta's While

Versuchen Sie, folgende Formel von Vieta ohne unnötige Wiederholungen zu berechnen.

$$2 \bullet \frac{2}{\sqrt{2}} \bullet \frac{2}{\sqrt{2+\sqrt{2}}} \bullet \frac{2}{\sqrt{2+\sqrt{2+\sqrt{2}}}} \bullet \bullet \bullet$$

6.6 Gemischte arithmetische Ausdrücke und Zuweisungen

6.6.1 Heron von Alexandria

Berechmen Sie die Fläche eines Dreiecks! Alle drei Seiten sind gegeben. Mit $s = \frac{a+b+c}{2}$ ist

$$F = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

(Heronische Formel). a= 65536 - 1.5/128; b= a + 3.5/128; c= 64+1/256.0 . Nadelförmiges Dreieck.

Vergleichen Sie die Ergebnisse bei doppelter Genauigkeit mit denen bei einfacher Genauigkeit!

Eine Umformung dieser Formel für a>=b>=c ist:

F = $\sqrt{(a+(b+c))\,(c-(a-b))\,(c+(a-b))\,(a+(b-c))}/4$

Berechnen Sie auch mit dieser Formel den Flächeninhalt des nadelförmigen Dreiecks! An dem Term (c-(a-b)) kann man erkennen, ob a, b und c wirklich Seiten eines Dreiecks sein können.

6.6.2 Mimikry

Geben Sie folgende Formel zuerst bis i=10 aus! Dann lassen Sie das Programm seinen Benutzer fragen, was als nächstes kommt. Geben Sie dann das 11. Glied der Folge aus!

`ceiling`($\sqrt{e^{i-2}}$)

6.6.3 Seine Grenzen kennen

Lassen Sie vorsichtig nur mit Hilfe von Konstanten 13!/4 berechnen, einmal mit `integer`-Arithmetik, ein anderes mal mit `real`-Arithmetik. Eine kleine Änderung in der Reihenfolge der Rechnung dürfte not- wendig sein.

Weisen Sie beide Ausdrücke je einer `integer`-Variablen zu. Drucken Sie diese aus!
Haben die beiden Variablen den gleichen Wert? Wenn ja, wieso?

Subtrahieren Sie nun von beiden Ausdrücken 1, noch vor der Zuweisung! Drucken Sie wieder beide `integer`-Ergebnisse aus! Was fällt auf? Begründen Sie so genau wie möglich das Ergebnis.!

6.7 Interpretation und Ausführung gemischter arithmetischer Ausdrücke und Zuweisungen

6.7.1 Nachdenken über die p-q-Formel

Diese Aufgabe ist als Beispiel in den Kursfolien verwendet.
Berechnen Sie die Lösungen einer quadratischen Gleichung

```
A * X**2 + B * X + C = 0 ( 1 )
```

Für den Fall " B**2 $>>$ A*C " gibt es unter Umständen einen großen Verlust an Genauigkeit, der durch eine algebraische Umformung vermieden werden kann (Wurzel in den Nenner). Vergleichen Sie die Güte aller (vier) Ergebnisse durch Einsetzen in (1)!
Beispiele: A=1, B=4097, C=0.25; A=1, B=-1.0E6, C=1; A=1.0E30, B=-1.0E36, C=1.0E30 .

6.8 Vor den Toren von Fortran: IEEE-Gleitkommazahlen bei der RS/6000

6.8.1 Eingeweideschau (etruskisches Orakel)

Geben Sie folgende Werte hexadezimal, oktal und binär aus (Fortran, nicht F):

1.0

0.1

tiny(1.0)*2.0**i mit i=15,-15,-1.

huge(1.0)

6.8.2 Berührung

Von wo bis wo geht die Nullstelle des Polynoms

```
f = -0.125e-30_p + x * (0.75e-30_p - x * (1.5e-30_p - 1.0e-30_p * x ))
```

Hinweis: Berechnen Sie mit Hilfe von binärem Suchen die Nullstelle einmal so, dass 0 als Funktionswert wie ein negativer Wert behandelt wird, ein zweites mal so, dass 0 wie ein positiver Wert behandelt wird.

Bei den Vergleichen der Funktionswerte f mit 0 können Sie entweder wirklich "$>$ 0", "$<$ 0"schreiben oder "$>$ tiny(f)", "$<$ -tiny(f)". Probieren Sie beides aus!

Geben Sie beim xlf-Compiler einmal *-qfloat=rndsngl* an und Optionen zum Finden von Underflows! Benutzen Sie den Debugger `xldb`!

6.8.3 Beinahe

Ist $e^{\pi \cdot \sqrt{163}}$ eine ganze Zahl? Rechnen Sie sehr genau!

6.8.4 In der komplexen Ebene

1. Wie berechnet man in einem Fortran-Programm aus einem gegebenen komplexen z (=x+i*y) entsprechend der mathematischen Formel "z = r*(cos(ϕ)+i*sin(ϕ))" die Anteile r und ϕ ? Diese erste Frage hat noch nichts mit IEEE zu tun.

2. Teilen Sie die normalgenaue komplexe Zahl z1=(a+i b) durch z2=(c+i d), indem Sie die Formel aus der Analysis benutzen. Was geschieht, wenn a=1e19, b=1e18, c=1e20 und d=1e19 ?

Benutzen Sie bitte einmal *xlf95* ohne Compiler-Optionen, ein andermal *xlf95 -qfloat=nofold:rndsngl* oder *xlf95 -g qflttrap=ov:inv:en -qsigtrap*. Testen Sie auch mit dem nagf95-Compiler!

Gibt es einen Unterschied zwischen z1*conjg(z2) / (real(z2)**2+aimag(z2)**2) und z1*conjg(z2) / r , wo die reelle Variable r=real(z2)**2+aimag(z2)**2 gesetzt wurde?

Formen Sie die Formel um: Wenn $|c|>|d|$ ist, teilen Sie conjg(z2) durch c (also 1- i d/c) und ebenso den Nenner (also c+(d/c)*d). Wenn $|d|>=|c|$, teilen Sie beide durch d.

Vergleichen Sie das nun erhaltene Ergebnis mit dem, was Fortran bei z1/z2 herausbekommt!

6.8.5 Erlaubt ist, was gefällt

Lassen Sie X^2-Y^2 mit `integer`- und Real-Arithmetik berechnen, beispielsweise mit X=4097 und Y=4096! Diese beiden Zahlen liegen bei der RS/6000 an einer kritischen Grenze. Was geschieht, wenn Sie erst u=x**2 und v=y**2 berechnen, dann u-v ?

Ändern Sie dann den Ausdruck mit Hilfe der binomischen Formeln und lassen ihn nun noch einmal direkt und nach Zwischenspeichern berechnen!

Versuchen Sie die Ergebnisse präzise zu begründen.

6.9 Ein/Ausgabe reeller und komplexer Zahlen: Formate

6.9.1 Ein Experiment mit Format

Variieren Sie bei `esw.d` das d zwischen (Precision(*rr_rp*)-1) und (Precision(*rr_rp*)+5) und dazu passend das w! Geben Sie mit diesen verschiedenen Formaten 10000 Zufallszahlen *rr_rp* aus und lesen sie gleich wieder in eine zweite Variable mit gleichem Typparameter ein!

Von welchem d an ist die Ausgabe immer mit der Eingabe identisch?

Versuchen Sie das alles bei einstellbarer Genauigkeit: Eine Änderung des Typparameters *rp* ist ausreichend, um das Programm zu variieren!

Erzeugen Sie die Zufallszahlen mit folgendem Aufruf:

```
call random_number(rr_rp).
```

6.9.2 Papiermaß

Schreiben Sie ein Programm zum Berechnen und übersichtlichen Ausgeben der A0- bis A6-Papiergrößen. Die Berechnung (in Metern) erfolgt nach folgender Formel:

$$langeSeite = 2^{\left(\frac{1}{4}-\frac{n}{2}\right)}$$
$$kurzeSeite = 2^{\left(-\frac{1}{4}-\frac{n}{2}\right)}$$

Die Ausgabe soll in Zentimetern erfolgen..

Kapitel 7

Ein/Ausgabe mit Namenslisten

7.1 Namelist

7.1.1 Abhängige Tabelle

Tabellieren Sie die Funktion

$$a\, e^{bx} + c\, e^{dx}$$

Verschiedene a, b, c und d sollen mit Namelist /fall/ gelesen werden. Dabei sollen nur Fälle akzeptiert werden, bei denen ac $>$ 0 und bd $<$ 0 ist. Die Tabelle soll 50 Werte umfassen von x = -2|g|bis x=2|g|mit

$$g = \frac{1}{d-b} \ln\left(\frac{-ab}{cd}\right)$$

Schreiben Sie die Namelist-Eingabe in einer ersten Version mit Ihrem Editor in eine Datei mit dem Namen fort.1! Dann schreiben Sie im Programm:

```
use system_ !nur fuer nagf95,sonst nicht
namelist /fall/ ...
call system('emacs fort.1')
open (1)  ! noetig nur bei nagf95
read(unit=1,fmt=fall)
```

Kapitel 8

Programmeinheiten

8.1 Anweisungsfunktionen, Formalparameter

8.1.1 Keine runde Sache

Es wird eine Anweisungsfunktion `sins(x)` gesucht mit der Eigenschaft `sins(x)=abs(sin(x))` für positive A, `sins(x)=-abs(sin(x))` für negative A. Dabei ist A eine Größe des Programms. Tabellieren Sie die Funktion von -2π bis 2π !

8.2 Definitionen von Funktionen

8.2.1 Nicht elementar: Integralsinus

Schreiben Sie eine Funktion *sini(x)*, die

$$\int_0^x \frac{sin\,(t)}{t} dt$$

durch gliedweise Integration der Reihenentwicklung von *sin(t)/t* berechnet.

$$\frac{sin\,(t)}{t} = 1 - \frac{t^2}{3!} + \frac{t^4}{5!} - + \ldots$$

Im Bereich $|x| \geq 20$ soll nicht gerechnet werden. Als Ergebnis soll dann *huge(x)* herauskommen, die Rechnung aber nicht abgebrochen werden.

Zum Vergleich kann man die doppeltgenaue Funktion `s13adf(x,idummy)` von der NAG-Library nehmen: beim Compilieren ”-l nag”angeben!

8.2.2 Rational denken

Ein Programm zur Rationalarithmetik.

Eingelesen werden zwei ganze Zahlen (Zähler und Nenner des Bruches 1), dann ein Operator (+,-,*,:), dann wieder zwei Zahlen (Zähler und Nenner des Bruches 2).

Es soll die angegebene Operation zwischen den Brüchen ausgeführt werden. Der Ergebnisbruch soll teilerfremd sein (denken Sie an Euklid!). Fixedoverflows sollen durch Kürzen möglichst vermieden werden.

Es muß an mehreren Stellen gekürzt bzw. das kleinste gemeinsame Vielfache berechnet werden.

Warum wurde in der Aufgabenstellung als Divisionsoperator ein : genommen und kein / ?

8.3 Definition von Subroutinen

8.3.1 Was ich nicht weiß, macht mich nicht heiß

Schreiben Sie zum Testen eine Subroutine, die zwei reelle Formalparameter hat, denen im Unterprogramm je der Wert 1.0 mitgegeben wird. Sonst nichts.

Übersetzen Sie das Programm getrennt vom Hauptprogramm!

Rufen Sie das Unterprogramm mit zwei Variablen als Aktualparametern auf, von denen eine - richtig - mit `real`, die andere aber - falsch- mit real(kind=dp) vereinbart ist, wobei dp=2*precision(1.0) ist. Übersetzung mit „xlf95 [-qextchk -bloadmap:loadmap] haupt.f90 sub.o“, Ausführung.

Was geschieht?

Legen Sie nun noch das Unterprogramm in dieselbe Quellprogrammdatei wie das Hauptprogramm (hintenangehängt). Übersetzen Sie beide zusammen (xlf95 und nagf95, bldmake+make [debug])! Was geschieht nun? Wenn der Compiler einen Fehler meldet, so ist das ein Entgegenkommen; er braucht das nicht zu tun.

8.4 Formalparameter-Unterprogramme, die `intrinsic`- und `external`-Anweisung

8.4.1 Stichproben

Eine Subroutine *minsuch(fun,a,b,n,xmin)* soll durch n-maliges Versuchen mit Hilfe des Zufallszahlengenerators *call random_number(zufallszahl)* die ungefähre Lage eines Minimums *xmin* der Funktion *fun* im Intervall *[a,b]* finden. $0 \leq$ *zufallszahl* < 1 , gleichmäßig gestreut.

Man suche beispielsweise in [-1,1] das Minimum von $e^{\frac{x}{2}} + e^{-x}$!

8.4.2 Ein Fall aus der Programmberatung

Folgendes stand in einem Benutzerprogramm:

```
implicit none
integer :: index
external [::] index
write(unit=*,fmt=*) index('Superlength','perle')
```

Was geschieht? Wieso? Beim xlf95- und beim nagf95-Compiler? Bei Cray kommt 3 heraus! Der F-Compiler dürfte `external` nicht akzeptieren.

8.4.3 Schlecht integriert

Schreiben und testen Sie eine Funktion zum Integrieren simpson(a,b,fun): *fun* soll von a bis b integriert werden. Das Hauptprogramm soll als *fun* `log` angeben, a=1 und b=e. Es sollte sich dann als Ergebnis 1 einstellen. Die Parameter sollen normal reell sein.

Das Ergebnis soll mit der Simpson-Regel gewonnen werden, die (besonders für rechte Deutsche) auch Keplersche Faßregel heißt. Dabei wird das Intervall (a,b) nach folgendem Schema immer genauer unterteilt:

```
1               4               1
|---------------|---------------|
a       h               h       b

1       4       2       4       1
|-------|-------|-------|-------|
a   h       h       h       h   b

1   4   2   4   2   4   2   4   1
|---|---|---|---|---|---|---|---|
a h   h   h   h   h   h   h   h b
```

Die Unterteilung kann weiter fortgeführt werden. An den senkrechten Strichen wird die zu integrierende Funktion berechnet. Mit dem darüber stehenden Faktor multipliziert werden die Funktionswerte zu einer Summe zusammengefaßt. Diese wird mit h/3 multipliziert, um eine Näherung für das Integral zu bekommen; h ist der jeweilige Abstand zwischen zwei Ordinaten.

Ist I_k die Näherung der Stufe k, dann ist $\delta = (I_k - I_{k-1})/15$ eine Schätzung des Fehlers von I_k. Der Ausdruck $I_k + \delta$ ist eine Verbesserung des Ergebnisses im Vergleich zu I_k.

Versuchen Sie bitte, die Funktion *fun* an jeder Stelle nur einmal zu berechnen! Begrenzen Sie die Anzahl der Halbierungen auf, sagen wir mal, 10. Wenn bis dahin der geschätzte Fehler noch immer nicht klein genug ist, geben Sie „not a number“ als Ergebnis zurück!

8.5 common-Blöcke, Block Data

8.5.1 Eine Funktion mit allem Drum und dran

Schreiben Sie eine Funktion func(x) (z.B. $b\sqrt{1-\left(\frac{x}{a}\right)^2}$ mit b=5 und a=8) und dazu im gleichen Quellprogramm ein Block-Data-Unterprogramm zur Festlegung eines x-Intervalls und der Anzahl der äquidistanten Unterteilungspunkte in diesem Intervall in einem `common`-Block. Dieses Quellprogramm wird getrennt übersetzt.

Schreiben Sie davon gesondert ein Quell-Hauptprogramm, das unter Benutzung des `commom`-Blocks die Funktion `func` tabelliert!

Man soll die Funktion und ein Intervall leicht modifizieren können, ohne das Hauptprogramm ansehen zu müssen.

8.6 Lebensdauer von Datenobjekten, save

8.6.1 Long life

Schreiben und testen Sie eine Subroutine grasin(x,singrad) zur Berechnung des Sinus `singrad`, wenn `x` in Grad vorliegt. Der benötigte Umrechnungsfaktor zwischen Grad und Bogenmaß soll nur beim ersten Aufruf berechnet werden und dann immer vorhanden sein.

8.7 interne Funktionen und Subroutinen

8.7.1 It's not necessarely true

Eine Subroutine *trans(a,b)* soll nur das reelle *a* dem reellen *b* zuweisen. Dementsprechend soll bei *a* nur Eingabe und bei *b* nur Ausgabe zugelassen sein.

Dieses Unterprogramm soll (fälschlicherweise) mit *call trans (1.0_dp,2.0)* aufgerufen werden, also mit einer doppeltgenauen 1 .

Was sagt der Compiler dazu, wenn es ein internes Unterprogramm ist? Was bei einem externen Unterprogramm im selben Quellprogrammpaket? Was bei getrennter Übersetzung? Hilft make debug?

Drucken Sie 1.0_dp und 2.0 nach dem Aufruf aus! Bei Fortran 77 war 2.0 nicht mehr da!

8.8 Umgebungszuordnung (Host Association)

8.8.1 An die Umgebung angepaßt

Lesen Sie n (>0) ein und berechnen damit die folgende Weiterentwicklung der Formel von Stirling in einer internen Funktion! Das Ergebnis soll hier zuletzt ganzzahlig sein; der Real-Wert soll zu ganzen Zahlen hin gerundet werden!

$n! = (2\,\pi\,n)^{0.5}(n/e)^{n}(1+1/(12\,n)+1/(288\,n^{2})-139/(51840\,n^{3})\ -571/(2488320\,n^{4}) + O(1/n^{5}))$

O(...) gibt dabei die Ordnung des Fehlers an.

Die Zahlen π und e sollen aus dem umgebenden Hauptprogramm übernommen werden. Die Genauigkeit bei der Berechnung der Formel soll sich nach der kleineren Genauigkeit dieser beiden Zahlen richten.

Wie groß darf n dabei werden? Bei `integer`-, `real`-, `real(kind=dp)`-Ergebnis ? Vergleichen Sie die Ergebnisse mit den richtigen Werten, soweit Sie diese kennen!

Bei der Berechnung eines Polynom oder einer Potenzreihe sollte man das Hornerschema benutzen, also statt

A + B*x + C*x ^{2} + D*x ^{3} + E*x ^{4}

besser:

A + x*(B + x*(C + x*(D + x*E)))

8.9 Das Formalparameter-Attribut `optional`

8.9.1 Gewichtsklassen

Eine Integer-Funktion `norm( groesse, gewicht [,geschlecht] [,alter])` soll über Untergewicht (-1), Normalgewicht(0) oder Übergewicht (+1) Auskunft geben. Als Normalgewicht soll *groesse*-100 gelten. Bei Frauen soll eine Toleranz von -7% bis +4%, bei Männern von -5% bis +5% zugelassen werden. Wenn die Angabe über das Geschlecht fehlt, soll so wie bei Männern gerechnet werden. Wenn die Größe kleiner ist als 120 cm, oder das Alter unter 17, soll das Ergebnis `huge(1)` sein.

8.10 Schlüsselwort-Parameter

8.10.1 Nutze, was Du hast

Die Fläche F eines Dreiecks kann man aus verschiedenen Angaben berechnen.

1. Alle drei Seiten sind gegeben. Mit $s = \frac{a+b+c}{2}$ ist $F = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$ (Heronische Formel).

2. Eine Seite und die dazugehörige Höhe ist gegeben: $F = \frac{a h_a}{2}$ usw. .

3. Zwei Seiten und der davon eingeschlossene Winkel sind gegeben: $F = \frac{1}{2} ab \, sin\, \gamma$ usw. .

4. Eine Seite und zwei Winkel sind gegeben: $F = \frac{c^2 sin\, \alpha \, sin\, \beta}{2 \, sin\, \gamma}$ usw. .

Machen Sie daraus eine Funktion Dreiecksfläche ([a] [,b] [,c] [,ha] ... [,alpha], ...) !

8.11 externe Funktionen und Subroutinen

8.11.1 My first picture show

Tabellieren Sie für ϕ zwischen -π /2 und 3π /2 in Schritten von π /90 nebeneinander:

ϕ

$\chi = \pi /2 - |\phi - \pi /2|$

$\rho = \text{sign}(\chi)*(1-[1-\{\chi /(\pi /2)\}^{6}]^{1/6}) + 2$

$x = \rho *\cos(\phi)$

$y = -\rho *\sin(\phi)$

Versuchen Sie, die Funktion auch an den kritischen Spitzen möglichst genau zu tabellieren. Dazu muß man die Grenzen des Intervalls und die Aufteilung recht genau berechnen. Unter Umständen muß man an 3π /2 ein kleine Korrektur vornehmen.

Natürlich liegt der Wunsch nach Graphik bei einer solchen Aufgabe nahe. Ich will hier einen Weg dazu zeigen.

Starten Sie die Graphik zu Beginn dieses Programms mit

```
use grsoft_interface_block
call grstrt(35,8)
! GR-Software initialisieren
call grsclc(2.0,2.0,27.0,27.0)
! Bildfenster im DIN A3 Format in CM
call grsclv(-2.5,-3.0,2.5,2.0)
! zugeordneter X-Y-Wertebereich
call grmrks(0.6) ! Symbolgroesse
call grnwpn(2)  ! Farbe 2 rot
```

In der Tabellierschleife, wo auch X und Y gedruckt werden, setzen Sie folgende Anweisung ein:

```
call grjmps(X,Y,209)
!Symbol Nr. 209 an Stelle X,Y zeichnen
```

Vor dem END des Programms müssen Sie die Graphik noch ausgeben; gleichzeitig wird die Graphik-Umgebung aufgegeben:

```
call grend()
```

Dieses Programm übersetzt man mit *grf90* statt mit *xlf95* (die Libraries werden dazugegeben) oder mit *bldmake/make* (anklicken der GR-Software). Auf dem Rechner `aix.zam` kann man auch mit `grF` übersetzen; man bleibt dann unter F-Kontrolle.
Nach dem Starten des Programms erscheint Graphik, die man durch Mausklick beendet.

8.12 Modul: Spezifikationsteil und Modul-Unterprogrammteil

8.12.1 Fliegendreck

Schreiben Sie ein Unterprogramm **zufa(iganz,vert[,mittel])** in einem Modul! Mit **iganz** sollen Zufallszahlen gebildet werden : **iganz=**mod(**iganz***1387,2^{19}-1). Wenn **mittel** fehlt, soll so gerechnet

werden, als sei es 1. Nach der Zufallszahlenformel soll **mittel**-mal **iganz** neu bestimmt und alle diese so gebildeten Zahlen addiert werden. Die Summe wird in Gleitkomma-Arithmetik durch (**mittel***(2^{19}- 1)) geteilt. Dieser Wert liegt zwischen 0 und 1.

Lassen Sie nun in einer Schleife 400 Paare (vertx,verty) berechnen mit zwei verschiedenen Anfangszahlen (iganzx, iganzy). Auch **mittel** kann bei beiden Folgen verschieden sein. Lassen Sie die Punkte zeichnen wie in der Übungsaufgabe "my first picture show".

```
use grsoft_interface_block
call grstrt(35,8)
call grsclc(2.0,2.0,27.0,27.0)
call grsclv(0.0,0.0,1.0,1.0)
call grmrks(0.4) ! Symbolgroesse
call grnwpn(5)  ! Farbe
```

Das ist die Einleitung. Den einzelnen Punkt erzeugt man in der Schleife mit:

```
call grjmps(vertx,verty,-107) !
```

Als Variante kann man auch nur den ersten Punkt mit `grjmps` zeichnen, alle folgenden dann mit `grdrws`! Der Aufruf ist nicht anders.

Am Ende zum Abschluß und zur Ausgabe:

```
call grend()
```

Eine Bitte: Jede Gruppe sollte ein anderes **mittel** wählen.

8.13 `use`-Anweisung, `use`-Zuordnung

8.13.1 Die Grundschwingungen

Berechnen Sie zwischen - π und π an 129 äquidistanten Stützstellen x die folgende Funktion:

$$f\left(x\left[,n\right]\right) = \frac{8}{\pi^2}\left(\frac{1}{1^2}sin\,x\; - \frac{1}{3^2}sin\,3x\; + \frac{1}{5^2}sin\,5x\; - + \cdots \frac{1}{n^2}sin\,nx\right)$$

Diese Funktion soll als Modul-Unterprogramm geschrieben werden. Die Rechengenauigkeit in diesem Unterprogramm und im Hauptprogramm soll von einem gemeinsam benutzten Modul aus gesteuert werden, in dem auch andere gemeinsame Größen stehen können. Wenn *n* beim Aufruf angegeben wurde, soll die Reihe nur bis *n* (bzw.*n-1*) berechnet werden, in jedem Fall aber nur so lange, wie die Summe sich noch ändert.
Lassen Sie die Punkte wie folgt zeichnen:

```
use grsoft_interface_block
call grstrt(35,8)
call grsclc(2.0,4.0,37.5,25.0)
call grsclv(-pi,-1.0,pi,1.0)
call graxs(-1,'X=3,Y=3',-1,' ',-1,' ')
call grnwpn(5)  ! Farbe
```

Das ist die Einleitung. Den ersten Punkt erzeugt man mit:

```
call grjmp(x,y)
```

Alle folgenden Punkte mit:

```
call grdrw(x,y)
```

Am Ende zum Abschluß und zur Ausgabe des Bildes:

```
call grend()
```

Wegen des *USE grsoft_interface_block* muß der Aufruf eine GR-Programmes mit einem falschen Typ/Typparameter auffallen. Bitte überprüfen Sie diese Aussage!

8.13.2 Spielend lernen, was ein GUI ist.

Spielen Sie mit den Test-Programmen der Library *xforms*! Schreiben Sie ein Programm, das mit *xforms* etwas einliest und eine Fehlernachricht anzeigt, falls es z.B. negativ ist.

8.13.3 Eine Funktion mit allem Drum und dran (2)

Schreiben Sie eine Modul-Funktion `func(x)` (z.B. $b\sqrt{1-\left(\frac{x}{a}\right)^2}$ mit b=5 und a=8) und dazu im Spezifikationsteil die Festlegung eines x-Intervalls und der Anzahl der äquidistanten Unterteilungspunkte in diesem Intervall. Dieses Quellprogramm wird getrennt übersetzt.

Schreiben Sie davon gesondert ein Quell-Hauptprogramm, das unter Benutzung des Moduls die Funktion func tabelliert!

Man soll die Funktion und ein Intervall leicht modifizieren können, ohne das Hauptprogramm ansehen zu müssen.

8.14 Einfache Schnittstellendefinitionen

8.14.1 Ruhe oder Leerlauf?

Schreiben Sie eine Modul-Funktion **next**(i). Diese Funktion soll aus dem Eingangswert ihres Parameters den nächsten Wert einer Folge berechnen.

Eine ziemlich teuflische Folge dieser Art ist: $f_{i+1}=f_i/2$ wenn f_i gerade, $f_{i+1}=f_i*3+1$ sonst.

Dies ist ein Beispiel für eine Folge, die jedes Element nur aus dem vorigen bestimmt, und zwar immer auf die gleiche Art. Wenn in einer solchen Folge ein Wert zweimal vorkommt, ist sie vom ersten dieser Werte an periodisch.

Schreiben Sie in einem zweiten Modul eine Funktion **period(fun,start,imax)**, die ausgehend von **start** die Länge der Periode bestimmt, welche eine durch **fun** definierte Folge hat! Dies **fun** ist eine Funktion nach der Art von **next** . In **period** gibt es eine Schnittstellendefinition für **fun** .

Verfahren: In einer Schleife (**imax** Durchläufe) bestimmen wir f_i (**ein** neuer Folgenwert pro Durchlauf) und f_{2i} (**zwei** neue Folgenwerte pro Durchlauf). Wenn igendwann $f_i = f_{2i}$ ist, dann ist die Folge sicher von i an periodisch (i ist ein Vielfaches der Periode). Wir müssen nun noch, ausgehend von i, die wirkliche Periodenlänge bestimmen.

8.15 Schnittstellendefinition mit generischem Namen

8.15.1 Das richtige π

Schreiben Sie einen Modul und darin eine Funktion pi(x), die je nach der Genauigkeit von x das π dazu passend zurückgibt.

```
write(unit=*,fmt=*) pi(1.0),pi(1.0_dp),pi(1.0_qp)
```

8.15.2 Du weißt schon, was ich meine

Schreiben Sie eine generische Funktion
binomial(a,k) = {a*(a-1)*(a-2)*...*(a-k+1)}/k!
für normal-reelles und doppeltgenaues a!

Bei k= 0 soll das Ergebnis 1 sein. Bei $k<0$ soll stattdessen $|k|$ genommen, aber am Ende vom Ergebnis der Kehrwert gebildet werden.

Wenn a eine größere ganze Zahl ist (z.B. `binomial(49.0,6)`), sollten keine unnötigen Rundungsfehler auftreten.

8.15.3 Unglauben mit Library: Experto credite!

Überprüfen und klären Sie die Ergebnisse der Aufgabe *Ein altägyptischer Brunnen*! Rufen Sie dazu das Programm *nag_polynom_roots (a,z)* der NAG-fl90-Library auf (Information: JSC-TKI-284, nagdoc)! Der Parameter *a* enthält die Koeffizienten. Er kann reell oder komplex sein, einfachgenau oder doppelt genau. *Z* ist komplex mit dem gleichen Typparameter.

Um die notwendige explizite Schnittstelle herzustellen, ist
use nag_polynom_eqn, only:nag_polynom_roots
vonnöten.

8.16 Die Unterprogramm-Präfixe `pure` und `elemental`

8.16.1 Euklids Elemente

Schreiben Sie ggT als generische Funktion, mindestens für Skalare und Vektoren! Ebenfalls als elementweise Funktion (Fortran 95).

8.17 Rekursive Unterprogramme

8.17.1 Euklid - wieder und wieder

Programmieren Sie den Euklidischen Algorithmus mit Hilfe einer rekursiven Funktion !

8.17.2 Zurück zum Einfachen!

Schreiben und testen Sie eine rekursive Funktion expo(x,n)! Sie soll x**n berechnen, indem sie den Ausdruck geschickt auf Multiplikationen zurückführt. x**7 => x*(x*x**2)**2.

Ist n=0, dann ist das Ergebnis 1. Was kann man tun, wenn n negativ ist? Wenn n gerade ist, soll expo(x,n/2) gerechnet werden. Was kann man tun, wenn n ungerade ist?

Natürlich kann ein Compiler bzw. eine Bibliotheksroutine mindestens genau so geschickt vorgehen, wie es diese Aufgabe vorschlägt - übrigens tun das keineswegs alle. Wir wollen es aber einmal selbst machen, um Rekursion zu lernen.

8.17.3 Ein Auffahrunfall

Berechnen Sie folgenden Kettenbruch 4/nenner(0,n) rekursiv:

$$\cfrac{4}{1+\cfrac{1}{3+\cfrac{4}{5+\cfrac{9}{7+\cfrac{16}{9+\cdots\frac{n^2}{2n+1}}}}}}$$

Nehmen Sie zum Testen 4/nenner(0,10)! Es ergibt sich eine wohlbekannte Zahl. Die rekursive Funktion **nenner** hat den Typ `real`.
Natürlich kann man einen Kettenbruch besser nach Art des Hornerschemas berechnen. Aber das soll hier nicht geübt werden.

8.17.4 Gelungen: die Quadratur des Kreises

Schreiben Sie eine Funktion zur adaptiven Berechnung eines bestimmten Integrals
adapt_quad(f,a,b,tol)!

Dabei ist f eine Funktion, *a* und *b* sind die Grenzen.

Man berechne zuerst *ein* Trapez: t1 = (b-a)/2*(f(a)+f(b)). Dann berechnet man über demselben Grundintervall *zwei* Trapeze: t2 = (b-a)/4*(f(a)+2*f((a+b)/2)+f(b))

Nun betrachtet man die Differenz *d=t2- t1*. Wenn $|d|$ kleiner als *tol* ist, soll t2+d/3 als Ergebnis gelten.

Wenn $|d|$ zu groß ist, wird als Ergebnis folgendes genommen (man berechnet erst ein Integral bis zur Mitte des Intervalls, dann von der Mitte bis zum Ende):

adapt_quad(f,a,(a+b)/2,tol/2)+adapt_quad(f,(a+b)/2,b,tol/2)

Beispiel einer zu integrierenden Funktion:
$f(x) = \frac{2}{1+x^2}, a = -1, b = 1$. Ergebnis: π .

Ein weiteres Beispiel:
$f(x) = log(x), a = 1, b = e$. Ergebnis: 1.

8.17.5 Vor-Zurück-Rekursiv

Schreiben Sie ein Programm, das eine Datei mit ganzen Zahlen liest. In jeder Zeile wird nur das erste Wort als Zahl gelesen. Die Zahlen sollen im allgemeinen in derselben Reihenfolge wieder ausgegeben werden. Wenn aber eine ungerade Anzahl von negativen Zahlen gelesen wurde, sollen die Zahlen bis zur nächsten negativen Zahl (bzw. bis Dateiende) in umgekehrter Reihenfolge erscheinen. Beispiel (hier stehen die Zahlen nebeneinander):

1,2,3,-4,5,4,2,3,-6,-6,4,23,-7,0,2,7
1,2,3,-4,3,2,4,5,-6,-6,23,4,-7,0,2,7

Wenn das erste Wort einer Zeile nicht als Zahl lesbar ist, darf das Programm abbrechen. Es sollte aber die zugehörige Zeilennummer ausgeben.

Am Ende der Datei soll ausgegeben werden, welche Satznummer der Endfile-Satz haben würde.

Die Aufgabe soll hier durch Rekursion gelöst werden.

Weiteres Beispiel : Kult mit Namen (später).

Kapitel 9

Vordefinierte Datentypen (erweiternde Zusammenfassung)

9.1 Typvereinbarung, Regeln für Character

9.1.1 Wirrwarr

Was bedeutet die folgende Anweisung(nicht F):
`character*5, allocatable, pointer, dimension(10)`

9.1.2 Klone

Bei einer Zeichenfolge soll untersucht werden, ob manchmal gleiche Zeichen nebeneinander vorkommen. Es soll die Stelle gesucht werden, an der die meisten gleichen Zeichen nebeneinander stehen. Ausgabe: Wo beginnt diese Teilfolge? Welches Zeichen ist es? Wie lang ist diese Teilfolge?

Hinweis zu einer Lösung:
Wenn man von links nach rechts in der Zeichenfolge sich vorarbeitend an einer Position *i* unmittelbar nach einem Wechsel steht, und wenn die bis dahin längste Folge gleicher Zeichen p Zeichen hatte, dann kann bei *i* nur dann ein neuer Kandidat für ein längstes Stück beginnen, wenn das Zeichen in Position *i+p* mit dem Zeichen in Position *i* übereinstimmt.

9.1.3 Kult mit Namen

Schreiben Sie ein rekursives Unterprogramm anagram(c,p)! Dabei ist c eine Zeichenfolge und p eine Position innerhalb dieser Folge. Von p an sollen alle Permutationen der Folge gebildet werden. Wenn p>=len(c) ist, soll c geschrieben werden.

Testbeispiel: call anagram('beil',1).

9.1.4 Andere Umlaute

Eine Funktion mit einem Character-Formalparameter beliebiger Länge soll ein Character-Ergebnis von derselben Länge haben. Die Eingabe soll in einer einfachen Weise umgeformt werden. Wenn zum Beispiel
`Das R"atsel der t"onernen F"u"se.`
hineingegeben wird, soll
`Das Raetsel der toenernen Fuesse.`
herauskommen.

9.1.5 Verhext

Schreiben Sie eine generische Funktion *hex*, die Logical, Integer, Character, Real, Doubleprecision in hexadezimale Character umwandelt. (Auswahl: Integer,Character).

Benutzen Sie zum Umwandeln möglichst nicht interne Dateien, da dann z.B. *write(unit=*,fmt=*) hex(129)* nicht überall möglich ist! Einige Compiler sehen hier nämlich einen rekursiven Aufruf von Ein/Ausgabe-Routinen.

Kapitel 10

Felder

10.1 Feldarten

10.1.1 Darstellung als Kettenbruch

Verwandeln Sie n-stellige Gleitkommazahlen (z.B. π ,e) in Kettenbrüche! Ansatz:

$$r = j_0 + 1/\left(j_1 + 1/\left(j_2 + 1/\left(j_3 + ...\right)\right)\right)$$

Die j_i sind ganze Zahlen, die der Reihe nach in einem eindimensionalen Feld `vec` gespeichert und zur Kontrolle ausgedruckt werden sollen (z.B. bis i=m) . Man sollte `m` einlesen und das Feld entsprechend dimensioniert anlegen. Natürlich sollte m nicht viel größer als n sein.
Die Zahl *r* soll als Kettenbruch dargestellt werden. Man zerlegt sie in den ganzen Anteil j_0 und einen Restanteil (den Bruch). Von diesem bildet man den Reziprokwert, mit dem man wie mit *r* verfährt. Diesen ersten Teil sollte man mit einer Subroutine `to_fraction(r,vec)` lösen.
Dann soll aus den Koeffizienten j_i wieder die Gleitkommazahl zurückgewonnen werden. Man rechnet rückwärts wie beim Hornerschema. Dies löst man am besten mit Hilfe einer Funktion `from_fraction(vec)`.

10.1.2 Charakterisierung von Feldern

Klassifizieren Sie die folgenden Felder:

```
subroutine example (n,a,b)
  real,dimension(n,10) :: w
  real,dimension(:)    :: a
  real,dimension(0:)   :: b
  real, dimension(size(a)) :: work
  real ::d(n,n)
```

10.1.3 Lang ist's her

Zwei Tagesdaten in der sogenannten internationalen Form sollen eingelesen werden, z.B.

```
29 Aug 1939
 5 Apr 1977
```

Das Programm soll die Anzahl der Tage zwischen den beiden Daten ausgeben:
`Vom 29.08.1939 bis 05.04.1977 : nnn Tage`

Schalttage gibt es alle vier Jahre, nicht jedoch in den Jahren, die zwar durch 100 teilbar sind, nicht aber durch 400.

10.1.4 Koch-sche Kurve (Graphik)

Mit dieser Aufgabe soll auch die Benutzung einer Bibliothek(Library) geübt werden. Programme, welche die Graphik-Library GR/GR3 benutzen, kann man auf dem Ausbildungscluster ohne F mit `grf90` übersetzen und binden. Bei F muss man die Produktionsversion nehmen, die im Augenblick nur für die R50 (aix.zam) lizensiert ist; dort übersetzt man dann mit `grF`.

Schreiben Sie eine rekursive Subroutine `zacke(p1,p5,n)` !

P1 und p5 sind Punkte in der Ebene. Die gerichtete Strecke p1p5 soll durch die Punkte p2 und p4 in drei gleich lange Stücke zerlegt werden. Das mittlere Stück p2p4 wird fortgelassen und ersetzt durch die restlichen Seiten p2p3 und p3p4 des gleichseitigen Dreiecks, dessen (unsichtbare) Basis p2p4 ist.

Das Dreieck soll nach links gezeichnet werden.

Rufen Sie mit jedem der vier Teilstuecke wieder `zacke` auf, wobei n um 1 reduziert wird.

Vor dem Aufruf von `zacke` wird im Hauptprogramm `call grstrt(35,0)` eingegeben. Aus dem Hauptprogramm wird `zacke` mit n=5 gerufen, p1=(/1.0,1.0)/, p5=(/38.0,1.0/). Nach dem Aufruf von `zacke` im Hauptprogramm wird `call grend()` eingegeben, wodurch das Bild ausgegeben wird.

Wenn im Unterprogramm n==1 ist, wird nicht mehr erneut `zacke` aufgerufen, sondern der gerade berechnete Polygonzug mit p1,p2,p3,p4,p5 gezeichnet:

call grln(x_vec,y_vec,5).

Hier enthält x_vec die 5 x-Koordinaten, y_vec die 5 y-Koordinaten.

Hinweis zur Geometrie: Die Höhe des Dreiecks ist (sqrt(3.0)/2)*Seite. Ein Vektor kann durch Matrixmultiplikation mit

```
0 -1
1  0
```

um 90 Grad nach links gedreht werden.

10.1.5 Gamma (zwei Libraries)

Berechnen Sie an 50 Stellen, beginnend mit x(1)=0.1 und endend mit x(50)=5.0 die Gamma-Funktion der Nagfl90-Library.

Zeichnen Sie mit GR-Software einen Achsenrahmen (graxs), bei dem x von 0 bis 5.1 skaliert ist, y von 0 bis 25 (grsclv). Der Rahmen soll auf dem zugrunde liegenden A3-Zeichenfeld von (2.0,2.0) (linker unterer Eckpunkt) bis (27.0,27.0) (rechter oberer Eckpunkt) gezeichnet werden (grsclc).

Die Linie der Kurven soll dick und rot sein (grnwpn,grspts).

Die Reihenfolge der Graphik-Aufrufe muss wie folgt sein:

grstrt grsclc grsclv graxs grnwpn, grspts grln grend

Da sie zwei Libraries benutzen, müssen sie `grf90` und `nagfl90` kombinieren.
/usr/local/nagfl90/KFAbin und /usr/local/grsoft/LOCALbin .
KFAbin wird irgendwann in LOCALbin umgetauft.

10.1.6 Quick_Search

Programmieren Sie den Quick_Search-Algorithmus von Sunday, eine einfache und im Durchschnitt schnelle Variante des Boyer-Moore-Algorithmus. Es geht um die Aufgabe, ein bestimmtes Zei-

chenmuster `pat` (Länge *lpat*) in einem möglicherweise sehr langen Text `text` (Länge *ltext*) zu suchen.

Zur Veranschaulichung der Idee des Verfahrens erst ein Beispiel (die oberste Zeile ist der Text, `format` ist das Zeichenmuster):

```
Joh. 1.0 Im Anfang war die Information.
format
       format
              format
                format
                       format
                              format
```

An jeder der Positionen des Zeichenmusters (hier `format`) vergleicht man es mit der darüber befindlichen Zeichenfolge gleicher Länge innerhalb von `text`. Wieweit das Pattern `format` in jedem Schritt vorrückt, richtet sich nach dem Zeichen *bad_character* , das bei der augenblicklichen Position des Patterns diesem im Text unmittelbar folgt. Wenn dieses Zeichen, im ersten Schritt ein Punkt, nicht in dem Pattern vorkommt, kann das Pattern von Position *i* zur Position *i+lpat+1* vorrücken. Wenn das *bad_character* jedoch im Pattern vorkommt, darf man das Pattern nur um *lpat-last+1* verschieben, wobei *last* die Position des letzten Vorkommens dieses Zeichens in `pat` ist.

Es soll angenommen werden, dass der verwendete Zeichensatz 256 Zeichen umfasst (z.B. erweitertes ASCII). Verwenden Sie ein vor dem eigentlichen Suchvorgang berechnetes eindimensionales Feld *skip* mit Indizes von 0 bis 255 zur Bestimmung der Anzahl vorzurückender Positionen:
i = i + skip(Nummer des bad_characters im verwendeten Zeichensatz).
In *skip* ist also zu jedem möglichen Zeichen abgespeichert, wie weit das Zeichenmuster vorrücken darf, wenn dieses Zeichen das *bad_character* ist.

Schreiben Sie ein geeignetes Unterprogramm *qsearch(text, [ltext,] pat, [lpat,] matches,[maxmatches,] nmatches)*! Die Längen *ltext* und *lpat* der Strings sowie die Dimension *maxmatches* sollten im Unterprogramm sofort bekannt sein. Es hängt von der gewählten Programmiersprache ab, ob man sie dazu als Parameter übergeben muß. Es soll im Unterprogramm keine längenabhängige Zeit zum Feststellen der Längen verloren gehen.

matches ist ein eindimensionales Feld mit einer vom Benutzer vorgegebenen (geschätzten) Länge *maxmatches*. Es sollen hierin die ersten (bis zu *maxmatches*) Positionen von *pat* in *text* zurückgegeben werden. In *nmatches* steht die wirklich gefundene Anzahl der Vorkommen, falls diese kleiner oder gleich *maxmatches* ist. Wenn es mehr als *maxmatches* Vorkommen gibt, soll bei der Rückkehr aus dem Unterprogramm *nmatches = maxmatches+1* sein.

Wenn das gesuchte Muster gar nicht vorkommt, sollte das erkennbar sein. Was geschieht, wenn lpat>ltext ?

Hilfsmittel: die Umwandlung eines Zeichens in seine Nummer wird in Fortran durch *ichar(char_len_1)* bewirkt.

10.2 Handhabung allozierbarer Felder

10.2.1 Blaise Pascal

Versuchen Sie, das Pascalsche Dreieck in gefälliger Form auszugeben! Wieviele Zeilen es haben soll, wird eingelesen.

10.2.2 Auf engstem Raum

Einzulesen ist *n*. Ein Integer-Feld *a(n)* werde als langer Bitstring benutzt. Schreiben Sie eine Anweisungsfunktion `bltest(a,pos)`! Sie soll wie `btest` funktionieren, nur dass `pos` Werte bis `n*bit_size(a)-1` annehmen darf. Entsprechend wird `iblset` und `iblclr` gebraucht, sie sollen aber als Subroutinen geschrieben werden, nicht wie im Original als Funktionen.

10.2.3 Klausur96

Ein achsenparalleles Rechteck $gx(1) \leq x \leq gx(2)$, $gy(1) \leq y \leq gy(2)$ soll mit einem Raster von n*m gleichen Teilrechtecken überdeckt werden. Es sei z.B. n=30, m=78.

Diesem n*m-Raster wird eine reelle n*m-Matrix *mat* zugeordnet.

An den Mittelpunkten der Teilrechtecke soll eine reelle Funktion *fun*(x,y) berechnet werden, deren Wert dem zugeordneten Matrixelement zugewiesen wird.
Schreiben Sie zur Verwirklichung dieses Vorhabens eine Subroutine *fun_to_mat (fun,gx,gy,mat)* ! Der Funktionsname *fun* und die Grenzen *gx* und *gy* werden vom Hauptprogramm ans Unterprogramm übergeben. Die Matrix *mat* wird im Hauptprogramm angelegt, bekommt aber erst im Unterprogramm ihre Werte.

Das Hauptprogramm soll nun, damit ein sinnvolles Ganzes entsteht, nach dem Aufruf von *fun_to_mat* ein Programm zur Veranschaulichung der Funktion aufrufen: `call` *print_plot*(mat) - siehe Anhang weiter unten.
Die Funktion *ratfun* soll in einem separaten Modul stehen. Die Koeffizientenmatrix *A* soll dem Hauptprogramm zugänglich sein; dort soll *A* in Größe und Inhalt festgelegt werden.

lx = `lbound`(*A*,1); *ly* = `lbound`(*A*,2). *ux* = `ubound`(*A*,1); *uy* = `ubound`(*A*,2).

Als Beispielfunktion für*fun* soll eine rationale Summen-Funktion in zwei Veränderlichen benutzt werden:

$$ratfun\,(x,y) = \sum_{j=ly}^{uy} \sum_{i=lx}^{ux} A_{ij} x^i y^j$$

Anhang:

```
subroutine print_plot(mat)
! Das Programm rechnet die Werte von mat
! in kleine Buchstaben um
! (a-niedrigste Werte, z-hoechste Werte).
! Die Buchstaben-Matrix wird ausgedruckt
  real,intent(in),dimension(:,:)                    :: mat
  character(1),dimension( size(mat,1), size(mat,2)) :: plo
  real            :: mi,ma,sub26
  integer         :: i

  mi = minval(mat)
  ma = maxval(mat)
  sub26 = 26.0-3*spacing(26.0)
! Im ascii-code beginnen die Kleinbuch-
! staben an Position 97.
```

```
    plo = char( int( 97 + ((mat-mi)/(ma-mi))*sub26 ) )
    do i = 1, size(mat,1)
      write(unit=*,fmt=*) plo(i,:)
    end do
  end subroutine print_plot
```

10.3 Vordefinierte Feld-Abfragefunktionen

10.3.1 Mat_Mult

Ein Unterprogramm soll ein Feld A (Rang 2) und ein Feld B (Rang 1) multiplizieren und das Ergebnis auf Feld C zurückgeben. Wie könnten die Deklarationen dieser drei Felder aussehen?

10.3.2 Tridia

Schreiben Sie ein Modul-Unterprogramm tridia(a) zur Lösung eines tridiagonalen reellen Gleichungssystems mit n Unbekannten:

$$a_{i,i-1}x_{i-1} + a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i$$
$$a_{1,0}, a_{n,n+1} = 0$$
$$i = 1, .., n$$

Im Beispiel n=5:

```
A11*x1 +A12*x2                          =B1
A21*x1 +A22*x2 +A23*x3                  =B2
        A32*x2 +A33*x3 +A34*x4          =B3
                A43*x3 +A44*x4 +A45*x5=B4
                        A54*x4 +A55*x5=B5
```

Man berechnet einen Faktor, mit dem man die erste Zeile malnehmen müßte, damit der erste Term in der zweiten Zeile zu Null wird, wenn man die multiplizierte erste Zeile von der zweiten abzieht. Das Element, welches 0 werden soll, braucht man in Wirklichkeit gar nicht zu berechnen, wohl aber die anderen Elemente der zweiten Zeile.

Man zieht dann die geeignet multiplizierte zweite Zeile von der dritten ab und macht so weiter, bis zur letzten Zeile. Das letzte x kann nun unmittelbar berechnet werden; rückwärts (von unten nach oben) lassen sich danach alle Unbekannten bestimmen.

Sie dürfen annehmen, dass das Gleichungssystem ”gutmütig” ist, dass also keine Komplikationen auftreten, die das Verfahren behindern könnten.

Speichern Sie die Koeffizienten $a_{i,k}$, b_i sowie die Lösung x_k platzsparend auf einer (4,NMAX)-Matrix. Die Matrix a wird abgeändert, die X-Werte sollen auf a(4,i) zurckgegeben werden. Sollte die erste Dimension von a nicht 4 sein, dann setze man a(1,1) = INF und tue sonst nichts.

Testen Sie das Unterprogramm für kleine n ($n \leq 6$), und geben Sie die Eingabewerte $a_{i,k}$, b_i sowie die Lösung x_k (i,k=1,..,n) aus!

Eingabebeispiel:

```
5         / =n
0 10  1 12/ die 0 wird nicht gebraucht
1 18  1 40
2  5  3 31
3  7  2 47
1 12  0 64/ die 0 wird nicht gebraucht
```

10.4 Konstruierte Feldwerte (array constructors)

Siehe Aufgabe ”Revolution” und ”Das Nicht-DO pflegen” weiter unten!

10.4.1 Kurz und Klein

Es soll eine Modul-Funktion `lowercase(x)` geschrieben werden. Sie soll alle Grossbuchstaben in der Zeichenfolge x durch Kleinbuchstaben ersetzen. Alle anderen Zeichen bleiben, wie sie sind. Die so veränderte Zeichenfolge soll als Ergebnis y zurückgegeben werden. Es darf angenommen werden, dass die Zeichen in ASCII verschluesselt sind. Die Grossbuchstaben haben die Nummern 65 bis 90, die Kleinbuchstaben 97 bis 122. Die Umwandlung soll nach folgendem Muster erfolgen:
`y(j:j) = c( ichar( x(j:j) ) )`
Die Schwierigkeit der Aufgabe besteht darin, das hier verwendete konstante Feld c mit der Dimension (0:255) geschickt zu initialisieren.

10.5 Teilfelder (array sections) und Feldelemente

10.5.1 Unterteilungen

Gegeben sei die Feld-Deklaration
`real, dimension(50,20) :: a`

Beschreiben Sie die Teilfelder, die folgende Teilmengen von A beschreiben:

- die erste Zeile von A
- die letzte Spalte von A
- jedes zweite Element in jeder Zeile und Spalte
- jedes dritte Element jeder Zeile und Spalte in umgekehrter Reihenfolge
- Ein Teilfeld mit Größe 0

10.5.2 Die Ordnung auf den Kopf gestellt

Schreiben Sie ohne Verwendung von `do`-Konstrukten eine Programmsequenz, die ein Feld mit 100 Elementen definiert und mit den Werten 1,..,100 initialisiert. Danach sollen zwei `integer`-Werte aus $\{1,..,100\}$ eingelesen und die Reihenfolge der Feldelemente in dem durch die beiden Werte festgelegten Bereich vertauscht werden.

10.5.3 Think parallel

Schreiben Sie in einem Module *toolbox* eine generische Subroutine *count_up(x)* ! Dabei ist *x* ein Vektor vom Typ `integer`, `real` oder `real(kind=dp)` . *X(:)* soll Ausgabeparameter mit den

Werten $x_i = i - 1$ sein.

Die Besetzung mit Werten soll mit einem parallelen Algorithmus geschehen:
Die ersten *minv* (eine Namenskonstante, z.B. 8) Elemente werden sequentiell mit Hilfe eines Feldbildners erzeugt. Die weiteren Elemente erzeugt man nach folgendem Schema:

```
x(  minv+1:2*minv) = x(:  minv) +   minv
x(2*minv+1:4*minv) = x(:2*minv) + 2*minv
x(4*minv+1:8*minv) = x(:4*minv) + 4*minv
.    .    .
```

Am Ende muß noch ein Rest aufgefüllt werden.

10.5.4 Think further parallel

Es soll eine generische Subroutine `interab(x,a,b)` geschrieben werden. Die Parameter sind entweder alle vom Typ `real` oder alle vom Typ `real(kind=dp))` . *X* ist ein Vektor mit *n* Elementen. *A* und *b* sind skalare Eingabegrößen. *X* soll Werte zwischen *a* und *b* bekommen mit gleichem Abstand. Das erste Element von `x` soll wirklich `a` werden, das letzte `b`.

Berechnen Sie zunächst mit Hilfe von *count_up* einen Hilfsvektor $t_i = \frac{i-1}{n-1}$, dann $x_i = b \cdot t_i + a \cdot t_{n+1-i}$, alles ohne `do`-Schleifen oder Ähnlichem, nur mit Feldausdrücken. Kann man statt *t* auch *x* selbst nehmen?

10.5.5 Schach-Damen-Problem, rekursiv, Backtracking

Auf einem ”Schachbrett” mit n·n Feldern (n=4,5,...,10) sollen n Damen so postiert werden, dass sie sich nicht schlagen können.

Schreiben Sie ein Programm, das für jedes der obigen n die Anzahl der möglichen Stellungen bestimmt, und geben Sie für n=8 eine solche Stellung in geeigneter Form aus.
Das Programm wird im Kurs *Algorithmen und Strukturen* als Beispiel behandelt. Man braucht etwas Zeit, um den richtigen Ansatz zu finden.

10.6 Feldausdrücke und Feldzuweisungen

10.6.1 DIVIDE ET IMPERA

Versuchen Sie, folgende Schleife für einen Vektorrechner so abzuändern, dass im Kern ein Stück offensichtlich vektoriell abgearbeitet werden kann:

```
real,dimension(110) :: a
integer             :: i
do i =11,110
  a(i) = a(i-10) + 1.0 ! Rekursion
end do
```

Natürlich kann so etwas z.B. der Compiler von CRAY auch von allein.

10.6.2 Endspurt

Ein lineares Gleichungssystem sei schon soweit umgeformt worden, dass die Koeffizentenmatrix *a* eine obere Dreiecksmatrix ist, wie in folgendem Testbeispiel:

```
9 x1 + 2 x2 + 0.5 x3 + 3 x4 = 58
       8 x2 + 1.5 x3 - 1 x4 = 22
                5  x3 + 3 x4 = 25
                       7 x4 = 35
```

Schreiben Sie eine Funktion *loesung(a,b)*, wo *b* die rechte Seite ist. *A* und *b* sollen, wie es sich bei einer guten Funktion gehört, nicht verändert werden.

Hinweis zum Algorithmus: Der erste Schritt kann sein: Die rechte Seite auf einem Hilfvektor *r* umspeichern; *loesung(n) = r(n)/a(n,n)*; dann von allen rechten Seiten oberhalb *r(n)* die entsprechenden Werte der letzten Spalte von a - multipliziert mit loesung(n) - abziehen.

10.6.3 Klausur 97

Schreiben Sie eine pure generische skalare Funktion *poly(a,x)* für 32-Bit-Integer, 64-Bit-Integer, Normal-Real und Doppelt-genau-Real.
Die Funktion soll den Wert eines Polynoms bestimmen:

$$a_0 + a_1 x^1 + a_2 x^2 + ... + a_n x^n$$

Die *a*'s für ein reelles Beispiel (13 Dezimalstellen) mit *n*=11 sollen aus folgender Formel gewonnen werden:

$$a_i = \frac{47 + 151i + 120i^2}{15 + 194i + 712i^2 + 1024i^3 + 512i^4} \cdot \frac{1}{16^i}$$

Zur Berechnung von Zähler und Nenner des Bruches kann man wieder *Poly* verwenden (mit Integer-Parametern).

Das reelle Polynom 11-ten Grades soll für *x*=1 berechnet werden, aber nicht nach der ersten Formel mit den Potenzen von x. Für kleinere *n* ($n < nlim$, mit z.B. *nlim*=8) soll vielmehr nach dem Hornerschema verfahren werden:

$$a_0 + (a_1 + (a_2 + ... (a_{n-1} + a_n x)\, x ...)\, x)\, x$$

Ist jedoch der Grad größer oder gleich *nlim*, soll folgender paralleler Vektor-Reduktions-Algorithmus verwendet werden.
Solange $n>0$ ist, rechnet man:

$$1.\ a_{n+1} = 0$$

$$\begin{aligned} 2.\ a_0 &= a_0 + a_1 x \\ a_1 &= a_2 + a_3 x \\ &... \\ a_{[n/2]} &= a_{2[n/2]} + a_{2[n/2]+1} x \end{aligned}$$

$$3.\ n = [n/2]$$

$$4.\ x = x^2$$

Schritt 2 soll eine einzige Vektoroperation sein. Das Ergebnis steht am Ende in a_0. Natürlich darf man für den zweiten Algorithmus nicht das originale Parameterfeld *a* verwenden.

10.7 `character`-Felder

10.7.1 Teilfelder

```
character(len=10), dimension(0:5,3) :: c
```

Welche der folgenden Beschreibungen von Teilfeldern von C sind unkorrekt (und wenn ja, warum?)

```
C(2,3),C(6,2),C(0,3),C(4,3)(:)
C(5)(2:3),C(5,3)(9),C(4:3)(2,1)
C(5,3)(9:9),C(2,1)(4:8),C(3,2)(0:9)
C(5:6),C(,)
```

10.8 Anordnung von Feldern im Speicher

10.8.1 Elemente

Geben Sie jeweils das erste, zehnte, elfte und letzte Element der folgenden Felder an:

- `real,dimension(11) ::A`
- `real,dimension(0:11) ::B`
- `real,dimension(-11,0) ::C`
- `real,dimension(10,10) ::D`
- `real,dimension(5,9) ::E`
- `real,dimension(5,0:1,4)::F`

10.8.2 Position eines Feldelementes

```
integer, dimension (0:4,-4:7,4:12) :: a
```

Schreiben Sie ein Programm, das jeweils eine Liste i,j,k von drei Indexwerten einliest und dann die Position des Elements A(i,j,k) im Feld A ermittelt.

Zum Beispiel hat a(0,-4,4) die Position 1, a(0,-3,4) die Position 6

10.8.3 Indexraum

Ein dreidimensionales Integer-Feld *cub(l,m,n)* mit Dimensionsgrenzen <= 9 soll im Hauptprogramm angelegt und zunächst mit Nullen besetzt werden.

1. Es soll zur später benötigten Kontrallausgabe ein Unterprogramm *cub_aus(cub,k)* mit expliziter Schnittstelle geschrieben werden, das dieses Feld ebenenweise ausgibt, wobei *k* (1 <= *k* <= 3) angibt, welcher Index zur zweidimensionalen Ausgabe einer Ebene auf Papier nicht verwendet wird - ist *k*=2 , dann werden also hintereinander *m* Ebenen der Größe *l***n* ausgegeben.

2. Es soll ein Unterprogramm *indcub_assh* geschrieben werden, das Feldübergabe mit übernommener Gestalt benutzt, und ein Unterprogramm *indcub_exsh*, das die alte Art der Feldübergabe mit expliziten Grenzen verwendet. Beide Programme sollen das gleiche tun:

Die Elemente eines übergebenes Teilfeldes *part* von *cub* sollen durch *part(i,j,k) = 100*i+10*j+k* einen Wert erhalten, wobei *i, j* und *k* jeweils von 1 an bis zur zugehörigen oberen Indexgrenze des Teilfeldes laufen.

Helfender Hinweis: Die beiden folgenden Aufrufe bewirken dasselbe

```
call indcub_assh(cub)
call indcub_assh(cub(:l,:m,:n))
! Versuchen Sie write(unit=*,fmt=*) cub
```

Ebenso

```
call indcub_exsh(cub,l,m,l,m,n)
call indcub_exsh(cub(1,1,1),l,m,l,m,n)
! Versuchen Sie write(unit=*,fmt=*) cub
```

3. Es soll ein Unterprogramm *lincub_assh* geschrieben werden, das Feldübergabe mit übernommener Gestalt benutzt, und ein Unterprogramm *lincub_exsh*, das die alte Art der Feldübergabe mit expliziten Grenzen verwendet. Beide Programme sollen das gleiche tun:
Die Elemente eines übergebenes Teilfeldes *part* von *cub* sollen als Wert die Nummer des Elementes erhalten, welche das Element bei linearer Speicherung des Teilfeldes haben würde.

10.8.4 Elemente eines Teilfeldes

Gegeben sei ein 2-dimensionaler Bereich ganzer Zahlen von 10 Zeilen und 10 Spalten.

In einem Hauptprogramm werden 10 Zahlenpaare (n,m) mit $1 \leq$ m,n≤ 10 eingelesen. n und m seien hierbei die Zeilen- bzw Spaltenanzahl von Teilfeldern in dem o.a. Bereich.

Schreiben Sie ein Unterprogramm, das für solche n*m-Bereiche in jedes Feldelement die Ordnungsnummer der Speicherfolge einträgt. Im rufenden Programm werden die so besetzten Felder zeilenweise ausgegeben. Benutzen Sie beim Aufruf des Unterprogramms auch Anfangsadressen wie A(2,3) oder A(7,1)

10.9 `random_number, random_seed`: Erzeugen von Pseudo-Zufallszahlen

10.9.1 Ziehung der Lottozahlen

Schreiben Sie eine Subroutine *ziehe(nv,m)*! Dabei ist nv ein Integer-Vektor der Länge n (z.B. 6). Es sollen aus den Zahlen 1 bis m (z.B. 49) zufällig n Zahlen gezogen und auf nv zurückgegeben werden. Sollte m$<$n sein, dann soll so getan werden, als sei m=n.

Kann *ziehe* externe Subroutine sein? Wann? Interne? Modul-Subroutine? Programmieren Sie *ziehe* als Modul-Unterprogramm!

10.9.2 Wahl 98

Sie haben durch Los die Teilnahme an einem Gewinnspiel der Jülicher Werbegemeinschaft an der Haupttribüne der Laga gewonnen.

Es sind dort fünf große Vorhänge angebracht worden, in Schwarz, Rot, Gelb, Tiefrot und Grün. Hinter einem der Vorhänge steht ein elchgetesteter Zweisitzer, den Sie gewinnen können.

Sie sollen sich zunächst für eine Farbe entscheiden und wählen z.B. Schwarz.

Dann sagt Ihnen der Spielleiter, Sie dürften Ihre Entscheidung noch einmal ändern; dabei gibt er Ihnen eine Hilfe: er öffnet eine der vier von Ihnen nicht gewählten Vorhänge, z.B. Gelb, hinter dem kein Auto steht.

Ist es nun nützlich, Ihre Wahl zu revidieren, oder können sie ohne Schaden bei Ihrer ersten Wahl bleiben?

Versuchen Sie, diese Frage durch Simulation mit `random_number` zu lösen. Sie spielen die Situation n-mal durch und zählen die Gewinne bei beiden Strategien.

Der Zufallszahlengenerator, der zum xlf-Compiler gehört, hat die Eigenschaft, immer wieder mit derselben Folge von Zufallszahlen zu starten. Sorgen Sie mit Hilfe von `random_seed` dafür, dass ihr Programm bei mehrfachen Aufrufen die Folge der Zufallszahlen jeweils dort fortsetzt, wo sie beim vorigen Aufruf endete. Dazu muss man am Anfang des Programms eine permanente Datei, wenn es diese schon gibt, lesen (das sogenannte Seed). Am Ende des Programms schreibt man das aktuelle Seed auf diese Datei - der unter Umständen vorhandene alte Inhalt muss überschrieben werden.

```
inquire(file=dateiname,exist=schon_da) ! Vor dem Lesen
open(unit=number,file=dateiname,status="old", &
     action="read",form="unformatted")  ! Zum Lesen

open(unit=number,file=dateiname,status="replace",
     action="write",form="unformatted") ! Zum Schreiben
```

10.9.3 Probleme mit der Familie

Ausbreitung und Verschwinden von Familiennamen.

Gegeben seien in einem Ausgangszustand N Familiennamen (N z.B. 20). Zu jedem Namen ist abgespeichert, wie oft er vorkommt (also wieviele Familien mit diesem Namen es gibt). In einem ersten Versuch können Sie annehmen, dass jeder Name nur fünf mal vorkommt.

In der weiteren Beschreibung werden diejenigen Kinder, welche ihren Familiennamen an ihre Kinder weitergeben, Namenshalter genannt. In Europa waren das bisher meistens die Söhne. Scheidungen und andere Komplikationen sollen hier nicht berücksichtigt werden.

Wieviele Namenshalter, die auch heiraten und den Namen weitergeben können, gibt es in einer Familie? Es soll dazu folgende Tabelle von Wahrscheinlichkeiten benutzt werden. Oben steht die Anzahl der Namenshalter-Kinder, darunter die Wahrscheinlichkeit für diese Anzahl.

```
|0   |1   |2   |3   |4   |5   |6   |
|.317|.365|.209|.080|.023|.005|.001|
```

Zu jeder Familie erzeugen Sie eine Zufallszahl R zwischen 0.0 und 1.0; je nach R stirbt nun die Familie aus (R<=0.317), bleibt in der folgenden Generation mit gleicher Anzahl erhalten (0.317<R<=0.682),

verdoppelt ihre Anzahl (0.682<R<=0.891) usw.

Das Schicksal aller Familien mit einem bestimmten Namen zusammengefaßt entscheidet, wie oft dieser Name in der nächsten Generation vorkommt.

Betrachten Sie wenigstens 30 Generationen! Veranschaulichen Sie die Entwicklung, indem Sie für N=20 die Namenshäufigkeit als Tabelle ausgeben: nebeneinder die Häufigkeiten der 20 Familiennamen, untereinander die Generationen. Wenn irgendwo die Zahlen (für's Ausdrucken) zu groß werden sollten, darf die Tabelle abbrechen.

Benutzen Sie den Zufallszahlengenerator *random_number* von Fortran!

10.10 Selbstgeschriebene Funktionen mit Feldergebnis

10.10.1 Horner-Schema

Ein Programm soll ein *n* einlesen und dann auf einen Vektor der Länge *n*+1 die Koeffizienten *a* eines Polynoms *n*-ten Grades.

Schreiben Sie eine generische Modul-Funktion *horner(a,x)*, die mit dem Horner-Schema den Wert eines Polynoms an der Stelle *x* berechnet. Das Polynom sei beschrieben durch das Feld *a* mit den *n*+1 Koeffizienten, wobei a_i Koffizient von x^i sei.
Man kann das Hornerschema rekursiv programmieren; das ist aber nicht Ziel der Aufgabe.

Die Funktion *horner* soll mindestens vier Fälle lösen: *x* zusammen mit *a* darf einfachgenau und doppeltgenau sein, *x* skalar und ein Vektor.

Frage am Rande: Könnte man *horner* auch als ELEMENTAL-Funktion schreiben?

10.10.2 Sort by Pack

Es soll eine rekursive Funktion *packsort(x)* geschrieben werden. *X* und *packsort* sind Vektoren gleicher Länge (*n*). Das Ergebnis (*ps*) der Funktion enthält die Elemente von *x*, aufsteigend sortiert.
Das Sortieren soll auf folgende Art zustande kommen:

Bei *n=0* muß die Funktion sofort verlassen werden.

Sonst setzt man *ps(1)=x(1)*. Bei *n=1* ist damit schon alles getan.

Sonst sorgt man nun dafür, dass das das erste und letzte Element von *ps* mit den entsprechenden Werten von *x*, aber in richtiger Reihenfolge, besetzt wird. Bei *n=2* ist damit die Arbeit getan.

Man bildet nun bei größerem *n* den mittleren Index *m = (n+1)/2* und fügt *x(m)* - unter Verwendung der schon vorhandenen *ps(1)* und *ps(n)* - so in *ps(1),ps(m),ps(n)* ein, dass diese Folge aufsteigend sortiert ist. Bei *n=3* ist damit die Funktion fertig.

Bei größerem *n* weist man dann *ps* einen Feldbildner mit drei Teilen zu: Der erste enthält alle Werte von $x < ps(m)$, die dann noch mit *packsort* sortiert worden sind. Der zweite enthält alle Werte von

x, die = *ps(m)* sind. Der dritte Teil enthält alle Werte von *x*, die > *ps(m)* und zusätzlich mit *packsort* sortiert sind.

10.10.3 Erstes Wahr

Schreiben Sie eine generische Funktion `firstloc` (*logic*). *logic* hat den Datentyp `logical` und ist möglicherweise ein Vektor, eine Matrix, ein Quader. Weiter brauchen Sie es nicht zu treiben. Im Falle eines Vektors soll das Ergebnis skalar sein, sonst ein Vektor.

Die Funktion soll bestimmen, welches Element des Feldes bei normaler Durchlaufreihenfolge als erstes .true. ist. Es gibt eine Ähnlichkeit zu `minloc` und `maxloc`.

10.10.4 Nachmachen

Die Funktion **minval** soll durch eine eigene Modul-Funktion *miva(fld[,dim])* nachgeahmt werden, die allerdings nur Vektoren und Matrizen „verstehen“ muß und keinen dritten Parameter *mask* hat.

Denken Sie an generische Schnittstellendefinitionen! Lösen Sie die Aufgabe „nach und nach“, zuerst den Vektorfall.

Zum Algorithmus: Er soll für einen Vektorrechner mit einer Vektorregisterlänge von 64 geschrieben werden. Das Feld *fld* wird in Stücke der Länge 64 unterteilt und ein Reststück (am Anfang oder am Ende). Es soll „parallel“ das Minimum gesucht werden, so dass man am Ende dieses ersten Schrittes einen Vektor der Länge 64 (oder weniger) hat, unter dessen Elementen das Minimum ist. Diesen Vektor kann man nun noch einmal auf 32 Elemente reduzieren, weiter auf 16 und 8. Von diesen wenigen Elementen suchen Sie mit einer `do`-Schleife das Minimum.

10.10.5 Fußball-Theorie

Einem Fußballspieler erscheint die Strecke *torb* zwischen den Pfosten des gegnerischen Tors von jeder Stelle (x,y) des Spielfeldes unter dem Winkel

```
w1=atan2(torb*x,x**2+y**2-torb**2/4)
```

Die Breite *torb* des Tores ist 7.32 m. Der Koordinatenursprung ist in der Mitte des Tores, die X-Achse ertreckt sich zum anderen Tor hin. Das Feld darf 9 *torb* breit sein und 13.5 *torb* lang.

Die Höhe des Tores ist *torb*/3. Vom Punkt (x,y) aus erscheint die Torhöhe unter dem Winkel

```
w2=atan2(torb/3 , sqrt(x**2 + y**2))
```

Der Raumwinkel, unter dem das Tor erscheint, ist dann (grob genähert)

```
w(x,y) = w1*w2*2/pi
```

Speichern Sie in eine 109*73-Matrix die Werte W(x,y). Jede Indexposition der Matrix entspricht einem Punkt auf dem Fußballplatz. Die beiden Achsen sollen äquidistant unterteilt werden.

Zeichnen Sie das Feld mit `grhhnl` oder `grhhfl`!
Muster: grdemo und gr3demo; `colhhl` und `filhnl`.

Da die Funktion auf dem größten Teil des Feldes ziemlich flach ist, sollte man die Abstände der Höhenlinien bei einer Verbesserung des Programms nicht-äquidistant bestimmen.

10.11 Vordefinierte Funktionen zur Feldreduktion

10.11.1 Ordne, aber nicht zuviel

Testen Sie folgendes primitive Sortierverfahren!

Eine Funktion `indup(x)` mit einem ganzen Vektor `x` von einem reellen Typ soll so berechnet werden, dass `x(indup(x))` aufsteigend sortiert ist.

Die Resultatvariable `indr` der Funktion ist also ein eindimensionales Integer-Feld von derselben Länge wie `x`.

Verfahren: von `x` wird eine Kopie `y` angelegt.
`indr(1) = minloc(y,1)` . Fortran95, aber unsere Fortran90-Compiler kennen es schon.
`y(indr(1)) = huge(y)`
.
.

10.11.2 Geld regiert die Welt

Gesucht ist ein Programm für einen Geldausgabeapparat. Der Benutzer gibt (mit maximal zwei Stellen hinter dem Komma) ein, wieviel DM (bzw. Euro) er haben will. Er kann auch noch bis zu drei besondere Wünsche äußern: wieviele Fünfhunderter, Zehner und Groschen (bzw. 10 Cent) er zum Beispiel haben möchte. Diese Sonderwünsche werden, wenn überhaupt möglich, primär erfüllt. Das noch verbleibende Geld wird so ausgegeben, dass es insgesamt möglichst wenige Scheine und Münzen ergibt.

Am Ende soll übersichtlich in wenigen Zeilen ausgedruckt werden, wieviel Geld von jeder Sorte herausgegeben wird, und wie die Sonderwünsche dabei berücksichtigt wurden.

10.11.3 Anna

Eine Funktion mit Logical-Ergebnis *palin(otto)* soll feststellen, ob die Zeichenfolge *otto* ein Palindrom ist, und zwar ohne `do`- oder `if`-Schleife.

10.11.4 Wer sucht der findet

Zu einer Funktion *fun* soll in einem Intervall [*a*,*b*] das Maximum gesucht und dabei ein Verfahren mit Zufallszahlen erprobt werden: *xmax* hat einen Wert aus [*a*,*b*], am Anfang z.B. (a+b)/2. Der dazugehörige Funktionswert sei *fmax*. Man bildet nun

$$xz = \left(\frac{\sum_{j=1}^{i} zuf_j}{i} - 0.5 \right) * 2$$

Die Zufallszahlen werden vor jeder Summenbildung neu erzeugt. Man berechnet
x = merge(xmax-a, b-xmax)*xz + xmax
und versucht, ob *fun(x)* größer als *fmax* ist. Falls das zutrifft, wählt man diesen Funktionswert als neues *fmax* und ändert auch *xmax* zu *x*.

I wird von 1 bis size (*nver*) erhöht, zu jedem *i* macht man *nver(i)* Versuche. Als *nver* gibt man einen Vektor mit 10 Elementen vor, jeder Wert ist 10. Dieser Vektor sollte vom Benutzer mit Hilfe eines Unterprogramms *set_nver* neu eingegeben werden können.

Der Benutzer ruft zur Bestimmung des Maximums
find_max(fun,a,b,xmax,fmax).
Als Testfunktion kann man z.B. *sin(x)-0.1*sin(11x)* nehmen, als Intervall *1.0* bis *2.0* . Alle Genauigkeiten sollten mit einem einzigen Typparameter änderbar sein, die Testfunktion aber nicht im gleichen Modul wie *find_max* und *set_nver* stehen.

10.12 Vordefinierte Funktionen mit Feldergebnis

10.12.1 Mit anderen Worten

Benutzen Sie `reshape` so, dass es dasselbe tut wie `transpose`!

10.12.2 Revolution

Es soll ein Ausdruck geschrieben werden, der bei einem Vektor mit 1:n Elementen das Stück 1:m mit dem Stück (m+1):n vertauscht.

Wenigstens zwei Lösungen !

10.12.3 Ein Bucket-Sort

Sortieren Sie in einer Modul-Subroutine *sort_int(iv)* ein Feld *iv* normaler Integer-Zahlen mit *bit_size(iv)* Bits auf folgende Weise:

Bringen Sie alle Zahlen in dem Vektor nach vorne, die als letztes (0-tes) Bit eine 0 haben, alle anderen nach hinten, unter sonstiger Beibehaltung ihrer Reihenfolge.

Tun Sie dasselbe mit dem vorletzten Bit, dann mit dem drittletzten usw. bis zum zweiten Bit.

Beim vordersten Bit müssen Sie gerade umgekehrt einsortieren: die 1-en nach vorne.

10.12.4 Non licet bovi

Das folgende Programm läuft mit dem nagf95-Compiler, aber nicht mit dem xlf95-Compiler. Auch nicht bei Cray. Zur Erklärung sei auf die Aufgabe *Eine Unzahl* ganz am Anfang der Aufgabensammlung verwiesen.

```
program bovi
  implicit none
  integer                  :: i
  integer,dimension(-5:5) :: ierg,iv,ik

  iv = (/(i,i=-5,5)/)
  ik = huge(1)
  ierg = merge( 60/iv, ik, iv/=0 )
  write(unit=*,fmt=*) ierg

end program bovi
```

10.12.5 Matrix-Multiplikation mit Schiebung

Die Matrixmultiplikation der linearen Algebra entspricht nicht der elementweisen Multiplikation bei Fortran 90. Bei n*n-Matrizen A,B,C ist die Bedeutung von C = A*B vielmehr :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj, i=1,...,n, j=1,...,n}$$

Der Cannon-Algorithmus dient dazu, unter n-maliger Verwendung der "parallelen"Fortran-Multiplikation zweier Hilfsmatrizen A1 und B1 das gewünschte Matrizenprodukt zu berechnen.

Die Elemente der Matrizen A1 und A2 werden vor jeder elementweisen Multiplikation derart angeordnet, dass jede Multiplikation ein neues Summenglied für jedes der Elemente c_{ij} der Produktmatrix berechnet.

Zunächst werden die Matrizen A1 und B1 in eine geeignete Ausgangsposition gebracht: Die Elemente von A1 ergeben sich aus den Elementen der Matrix A, indem jeweils die i-te Zeile der Matrix A um i-1 Positionen zirkular nach links geshiftet wird. Die Elemente der Matrix B1 ergeben sich aus den Elementen der Matrix B, indem jeweils die j-te Spalte von B um j-1 Positionen zirkular nach oben geshiftet wird.

Anschließend wird das elementweise Produkt von A1 und B1 gebildet und auf C abgespeichert. Damit ist für jedes Element von C bereits ein Summand berechnet.

Die restlichen n-1 Summanden erhält man durch wiederholtes Shiften der Matrizen A1 und B1 und anschließende elementweise Multiplikation. Dabei sind nun jeweils die Elemente der Zeilen der Matrix A1 um eine Position nach rechts und die Elemente der Spalten der Matrix B1 um eine Position nach oben zu shiften.

Beispiel, Ausgangsmatrizen (n=2):

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$A1 = \begin{pmatrix} a_{11} & a_{12} \\ a_{22} & a_{21} \end{pmatrix} \quad B1 = \begin{pmatrix} b_{11} & b_{22} \\ b_{21} & b_{12} \end{pmatrix}$$

Zwischensumme:

$$C = A1 * B1 = \begin{pmatrix} a_{11} * b_{11} & a_{12} * b_{22} \\ a_{22} * b_{21} & a_{21} * b_{12} \end{pmatrix}$$

Shiften der Zeilen von A1 um eine Position nach links und Shiften der Spalten von B1 um eine Position nach oben liefert:

$$A1 = \begin{pmatrix} a_{12} & a_{11} \\ a_{21} & a_{22} \end{pmatrix} \quad B1 = \begin{pmatrix} b_{21} & b_{12} \\ b_{11} & b_{22} \end{pmatrix}$$

Elementweise Multiplikation von A1 und B1 und anschließende Aufsummation auf C liefert die Produktmatrix:

$$C = C + A1 * B1 = \begin{pmatrix} a_{11} * b_{11} + a_{12} * b_{21} & a_{12} * b_{22} + a_{11} * b_{12} \\ a_{22} * b_{21} + a_{21} * b_{11} & a_{21} * b_{12} + a_{22} * b_{22} \end{pmatrix}$$

10.12.6 Das Nicht-Do pflegen

Zeichnen Sie im Rechteck mit der linken unteren Ecke (3.0,2.0) auf DIN A3 und der rechten oberen Ecke (37.5,26.7) die Funktion `atan(x)` mit dem Wertebereich für x von -2.1 bis 2.1 und y von -π /2 bis π /2 . Nehmen Sie zum Zeichnen der Funktion `grln` an z.B. 101 Stützstellen. Die beiden Felder x und y sollen ohne explizite `do`-Schleife berechnet werden.

Die Kurve soll grün sein (`grnwpn`) und etwas dicker als normal (`grspts`). Zeichnen Sie mit `graxs`(-1,'X=3,Y=3', -1,'X-Achse',-1,'Y-Achse') eine geeignete Achse und Beschriftung in weiß auf schwarz bzw. schwarz auf weiß.

10.13 `where`-Anweisung und `where`-Konstrukt

10.13.1 Wer hat, dem wird noch gegeben

Schreiben Sie eine `where`-Anweisung zum Verdoppeln aller positiven Werte eines Feldes!

10.13.2 Printplot

Bei einem Real-Feld *z(n,m)* sei jedem Indexpaar *(i,j)* ein reelles Wertepaar *(x,y)* zugeordnet:

x = (xmin+δ_x)*((n-i)/rn) + (xmax-δ_x)*((i-1)/rn)
y = (ymin+δ_y)*((m-j)/rm) + (ymax-δ_y)*((j-1)/rm)
mit $\delta_x = (xmax - xmin) / (2n)$ und
mit $\delta_y = (ymax - ymin) / (2m)$ und
mit rm = `real`(m-1) und
mit rn = `real`(n-1).

Man kann somit dem Element *z(i,j)* leicht einen Funktionswert *f(x,y)* zuweisen.

Für ein Beispiel nehmen Sie bitte *xmin,ymin*=-π und
xmax,ymax=π ! n=78, m=78; f(x,y) = sin(x)*sin(y).

Diese Matrix soll mit Hilfe von `forall` oder `spread` oder `matmul` berechnet werden! Rufen Sie mit einer solchen Matrix *putma* auf!

Schreiben Sie dieses Modul-Unterprogramm *putma*, dem die Real-Matrix *z* übergeben wird! In der Matrix werden die Funktionswerte von *f* übergeben. Der erste Index darf nicht größer als 78 (bzw. 132) werden. Wenn er größer sein sollte, werden im Programm nur 78 bzw. 132 davon genommen (Bildschirm-,Papiergröße).

Weiter wird *zmin* und *zmax* übergeben. Wenn es *z*-Werte gibt, die kleiner als *zmin* sind, dann sollen sie im folgenden wie *zmin* behandelt werden. Werte, die größer als *zmax* sind, werden wie *zmax* behandelt. *Z* soll aber erhalten bleiben.

Ordnen Sie den Z-Werten zwischen *zmin* und *zmax* ganze Zahlen zwischen zwischen 1 und 53 zu! Mit einer solchen ganzen Zahl pickt man sich einen Buchstaben aus dem String 'zyxw ... cba0ABC ... WXYZ' heraus. Alle diese Buchstaben einer Spalte von *z* werden aneinandergehängt und als Zeile ausgegen. Der erste Index von *z* wird also, wie üblich, der X-Achse zugeordnet.

Wenn man dies für die ganze Matrix macht, erhält man ein grobes Bild von z. Geben Sie alles so aus, dass die zweiten Indizes, die der Y-Achse zugeordnet sind, von unten nach oben zunehmen.

10.13.3 Schachbrett

Besetzen Sie die reelle 80*80-Matrix Z mit zwei Konstanten, so dass ein schachbrettartiges Muster entsteht.

Es müssen also 8*8 Teilfelder mit je 10*10 Indizes gebildet werden. Jedes dieser Teilfelder ist mit einer der beiden Konstanten besetzt. Die beiden Typen von Feldern sollen einander wie auf einem Schachbrett abwechseln. Jedes Teilfeld soll durch eine einzige Anweisung besetzt werden.

Gebt das ganze zur Kontrolle mit `putma` als Printplot aus!

10.14 `forall`-Anweisung und -Anweisungsgruppe (Fortran 95)

10.14.1 Hilbert

Berechnen Sie die n$\times$n-Hilbert-Matrix H(i,j)=m/(i+j-1), wobei m das kleinste gemeinsame Vielfache der Zahlen 1 ... n+n-1 ist.

Berechnen Sie zu dieser Hilbert-Matrix eine rechte Seite b, so dass x=(1,2,3, ... ,n) ein Lösungsvektor der linearen Gleichung H$\bullet$x=b sein sollte.

Schreiben und benutzen Sie ein Unterprogramm gauss(H,b,x,error) zum Lösen des linearen Gleichungssystems. Testen Sie mit n=3,7,10,13 bei Rechnung mit einer reellen Genauigkeit von `selected_real_kind(13)`!

Die Matrix ist sehr schlecht konditioniert, das heißt, schon bei etwas größerem n sind die Ergebnisse von *gauss* ziemlich falsch.

Kapitel 11

Ein/Ausgabe, externe Dateien

11.1 Beispiel: unbekannte Anzahl Zahlen/Zeile

11.1.1 Immer der Reihe nach

Eine Datei enthalte nur ganze Zahlen. Es ist nicht bekannt, wieviele jeweils in einer Zeile stehen, und wieviele Zeilen es gibt. Auch leere Zeilen können vorkommen. Die Zahlen stehen durch Leerzeichen oder Komma getrennt nebeneinander.

Ein Unterprogramm *Zahlensauger(uni,V,n)* soll von der Einheit *uni* von der augenblicklichen Position aus alle Zahlen der Reihe nach in das eindimensionale Feld *V* einlesen. Wenn *V* voll ist, soll das Unterprogramm mit *n=size(V)* beendet werden. Wenn der Lesevorgang am Ende der Datei angekommen ist, soll *n* die Anzahl der tatsächlich gelesenen Zahlen angeben; das Unterprogramm wird beendet.

11.2 Die open-Anweisung

11.2.1 Zeilensuche

Schreiben Sie eine Programmsequenz, die eine Datei zeilenweise einliest bis eine Zeile gelesen wird, die in den Spalten 3 bis 7 nur blanks enthält oder die mit einer Zahl beginnt. Überprüfen Sie zunächst, ob die Datei auch existiert!

11.2.2 Selbstdefinierte Dateien

Ein Programm soll mehrere unterschiedlich große zweidimensionale Feldbereiche (circa 100*100) im internen Format auf eine Datei A schreiben, jeden Feldbereich als Satz.

Die Laufbereiche der Indizes sollen selbst als erste Zahlen auf die Datei geschrieben werden (selbstdefinierende Sätze).
Beispiel:

write (...) ko,jo

write (...) feld2(:ko,:jo)

Schreibt ähnlich eine Datei B!

Nun soll ein zweites Programm Datei B an Datei A anhängen und B auslöschen.

11.3 Die `close`-Anweisung

11.3.1 Fenster zur Welt

Ein Hauptprogramm *erst* soll ein anderes mit Namen *zweit* starten, das in einem eigenen Fenster mit dem Benutzer interagiert:

```
call system('aixterm -T Input -e zweit')
```

Fordern Sie mit dem *Zweit*-Programm die Eingabe der geograpische Breite und Länge von zwei Orten an!
Beispiele:

```
Jülich       50.9  nord  6.35 ost
Mururoa      22.10  süd 138.5 west
Berlin       52.45 nord  13.4 ost
```

Das *Zweit*-Programm schreibt die Ortsnamen und die Koordinaten auf eine neue Datei *fort.1*. Das Programm *erst* soll daraus die Entfernung der Orte voneinander berechnen und ausgeben. Die Datei *fort.1* soll dann wieder entfernt werden. Als Entfernung nehmen wir die sogenannte "Luftlinie".

Zur Berechnung benutzen wir den Cosinussatz für sphärische Dreiecke

```
cos(e)=cos(90-b1)*cos(90-b2)+sin(90-b1)*&
       sin(90-b2)*cos(l1-l2)
mit
   b1,b2  : die Breite der beiden Orte
   l1,l2  : die Länge der beiden Orte
   e      : der Winkel zwischen den Orten
            auf dem Grosskrei
```

Der Erdradius ist 6371.03 km. Wir betrachten die Erde als Kugel.

11.3.2 Bilder auf Datei 1

Ein Programm soll auf eine Datei film.daten unformatiert schreiben. Wenn die Datei schon existiert, soll der bisherige Inhalt bleiben; die neuen Sätze sollen dann am Ende angefügt werden.

Der Inhalt der Sätze ergibt sich aus folgenden Angaben:

Es sollen reelle Felder des Typs F(3,N,M) erzeugt und gespeichert werden (Empfehlung: N=31, M= 9).

Es handelt sich um N*M-Matrizen mit drei Komponenten. Die drei Komponenten beschreiben einen Punkt im 3D-Raum. Das ganze Feld gibt die Transformation eines rechteckigen Gitters in den 3D-Raum wieder.

Für J=1 bis M nehme V gleichmäßig Werte zwischen -1 und 1 an. Für I=1 bis N nehme U gleichmäßig Werte zwischen -PI und PI an. Mit diesen U,V-Kombinationen und R=3 soll

```
F(1,I,J) = R*COS(U) + V*SIN(D*U)
F(2,I,J) =            V*COS(D*U)
F(3,I,J) = R*SIN(U)
```

berechnet werden. Insgesamt sollen 21 solcher Felder erzeugt werden: D = 0 bis 1 in Schritten von 0.05.

Das Feld F soll mit einem einzigen Schreibbefehl auf die Datei gebracht werden. Vor diesem Datensatz mit dem Feld soll immer ein kurzer Satz mit drei Integer-Zahlen geschrieben werden: eine 1 und N und M. An der 1 soll später erkannt werden, dass die Daten von diesem Programm stammen.

11.3.3 Bilder auf Datei 2

Das Programm soll auf die Datei film.daten (siehe Aufgabe 1) unformatiert schreiben. Wenn die Datei schon existiert, soll der bisherige Inhalt bleiben; die neuen Sätze sollen dann am Ende angefügt werden.

Der Inhalt der Sätze ergibt sich aus folgenden Angaben:

K1, K2, K3 seien drei kleine ganze Zahlen z.B. 2, 3, 5. Es sei durch Q(3,N) ein Linienzug im 3D beschrieben. N kann man 60*MAX(K1,K2,K3) bis etwa 120*MAX(K1,K2,K3) wählen. Wir benutzen folgende Größen:

```
OM1 = 2 * Pi * K1  ; PHA1 = 0.2 (z.B)
OM2 = 2 * Pi * K2  ; PHA2 = 1.4 (z.B)
OM3 = 2 * Pi * K3  ; PHA3 = 2.6 (z.B)
```

Für I=1 bis N nehme T gleichmäßig Werte zwischen 0 und 1 an. Damit berechnen wir:

```
Q(1,I) = SIN( OM1 * T + PHA1 )
Q(2,I) = SIN( OM2 * T + PHA2 )
Q(3,I) = SIN( OM3 * T + PHA3 )
```

Es handelt sich hier um eine 3D-Verallgemeinerung der Lissajous-Kurven.

Das Feld Q und eine skalare Größe R soll mit einem einzigen Schreibbefehl auf die Datei gebracht werden. Vor diesem Datensatz mit dem Feld und R soll immer ein kurzer Satz mit drei Integer-Zahlen geschrieben werden: eine 2 und N und M.

An der 2 soll später erkannt werden, dass die Daten von diesem Programm stammen.

M und R haben eine Bedeutung für die Darstellung des Linienzuges. Dieser soll später nicht nur als ganz dünne Linie im Raum gezeichnet werden, sondern als Schlauch mit einem (M-1)-eckigen

Profil. Das regelmäßige (M-1)-Eck soll einen Außenradius von R haben. Vernünftige Werte sind z.B. M=9 und R=0.025

Bitte bringen Sie etwa 10 solcher Datensatz-Paare auf die Datei! Dabei sollte R, (K1, K2, K3) und/oder (PHA1, PHA2, PHA3) variiert werden.

11.3.4 Ausgabe der Bilder

In dem Programm dieser Aufgabe sollen die Daten von Aufgabe 1 und 2 gelesen und graphisch dargestellt werden.

Sie sollten zuerst *export GRSOFT_DEVICE=210* als UNIX-Befehl eingeben. Das bedeutet: Ein Bild wird nach dem anderen ohne Eingriff erzeugt, und zwar nur am Bildschirm.

Im Deklarationsteil des Programms muß ein Feld FEIN vereinbart werden, in das eingelesen werden kann. Zusätzlich braucht man ein reelles Feld AR mit LAR Elementen. AR ist ein Arbeitspeicher für die Graphik-Routinen; AR sollte wenigstens 46 mal so viele Elemente wie FEIN haben.

Benutzen Sie „USE grsoft_interface_block“! Zu Beginn des ausführbaren Programms muß *grstrt(35,8)* aufgerufen werden; am Ende wird mit *grende* die Graphik beendet.

Es wird nun ein Datensatz-Paar nach dem anderen eingelesen. In jedem Fall wird mit

```
call gr3dim(LAR,IER)
```

ein 3D-Bild initialisiert. Nun folgt , wenn die erste Integer-Zahl 1 ist

```
call gr3net(AR,IER,N,FEIN,N,1,M,1,1,2)
```

Ist sie aber 2, dann steht hier stattdessen

```
call gr3tub(AR,IER,FEIN,N,M,R,.true.,1,6)
```

Nun folgt die nicht-interaktive Ausgabe der Graphik aufs Terminal:

```
call gr3rot(AR,IER,'Z',30.,'X',-60., &
            'Y',0.)
call gr3plo(AR,IER,'MIX')
oder    call gr3plo(AR,IER,'HID')
call grnxtf
```

11.4 Positionierung innerhalb sequentieller Dateien

11.4.1 Langer Atem

Lesen Sie die Datei **/home/fort116/f90texte/f90norm** von Anfang bis Ende und bestimmen dabei, welches der im Sinne von `len_trim` längste Satz ist. Drucken Sie ihn und seine Umgebung aus

(zwei vorher bis zwei nachher)!

11.5 Direkt-Zugriffs-Dateien

11.5.1 Alfs Sprüche

Es sollen zwei zusammengehörende Hauptprogramme geschrieben werden, die gemeisam einen getrennt übersetzten Modul benutzen, in dem die Satzlänge einer von beiden Programmen verwendeten Direkt-Zugriffs-Datei **dzd** steht.

Programm 1 liest (eventuell mehrmals) einen einzeiligen Spruch von der Standardeingabe. Wenn es **dzd** schon gibt, wird dieser Spruch hinten angehängt und die Satzanzahl, die im ersten Satz steht, um 1 erhöht. Wenn es **dzd** noch nicht gibt, wird die Datei angelegt: in Satz 1 wird die Zahl 1 oder 2 geschrieben (entweder die Anzahl der Sprüche oder der Sätze), in Satz 2 der erste Spruch.

Beim ersten mal wird auch mit `get` bei `random_seed` der erzeugende Zahlensatz des vordefinierten Zufallszahlengenerators hergeholt. Er wird dann auf eine sequentiellen Datei **szd** geschrieben.

Das zweite Programm soll mit Hilfe des vordefinierten Zufallszahlengenerators bei jedem Aufruf eine neue Zufallszahl erzeugen. Dazu muß **szd** gelesen und geschrieben werden. Man wählt ”zufällig”einen der Sprüche von der Datei und druckt ihn auf den Bildschirm.

Kapitel 12

Selbstdefinierte Datentypen und Operatoren

12.1 Komponentendefinition

12.1.1 ari

w sei ein Feld von n Zahlen. Schreiben Sie eine Funktion, die eine Struktur mit den beiden folgenden Komponenten zurückgibt:

- das arithmetische Mittel W der n Zahlen w_i
- den mittleren Fehler des Mittelwertes:

$$\sqrt{\left(\left\{\sum (w_i - W)^2\right\} / \{(n-1) * n\}\right)}$$

12.2 Konstruierte Strukturwerte

12.2.1 Bei Informatikern schafft man durch Mischen Ordnung

Schreiben Sie ein Modul, das eine Struktur TOWN definiert: Name einer Stadt, Lage (geographische Breite, Länge, Höhe über NN), Temperatur (Jahresminimum,-Maximum).

In dem Modul soll ein Unterprogramm zum ”Mischen” zweier Dateien liegen, deren Namen als Parameter übergeben werden. In jeder Zeile der Datei stehen die Daten einer Stadt. Innerhalb der Dateien seien sie schon aufsteigend nach Städtenamen sortiert. Es soll eine dritte Datei (der Name ist wieder Parameter) gebildet werden, in der alle Städte enthalten sind und in der auch wieder die Städtenamen aufsteigend sortiert sind.

12.3 Selbstdefinierte Operatoren

12.3.1 Rechnen mit Intervallen

Definieren Sie einen selbstdefinieten Typ Interval und die Rechenregeln für ihn!

[a,b] + [c,d] = [a+c,b+d]

[a,b] - [c,d] = [a-d, b-c]

[a,b] $\times$ [c,d] = [min(ac,ad,bc,bd),max(ac,ad,bc,bd)]

[ab] / [c,d] = [min(a/d,a/c,b/d,b/c),max(a/d,a/c,b/d,b/c)]

12.3.2 Quaternionen: mehr als komplex

Das Feld der Quaternionen a,b ... ist die einzige Erweiterung des Körpers der komplexen Zahlen, das eine nicht-kommutative Divisionsalgebra, basierend auf dem Körper der reellen Zahlen, bildet.

Jedes Quaternion a kann in folgender Form dargestellt werden:
$a = \alpha_0 + i\alpha_1 + j\alpha_2 + k\alpha_3$

Dabei sind α_0, α_1, α_2, α_3 reelle Zahlen, und i, j, k sind spezielle Quaternionen (Erzeuger) mit folgenden Multiplikationsregeln:
$i^2 = j^2 = k^2 = -1$
$jk = -kj = i \quad ; \quad ki = - ik = j \quad ; \quad ij = - ji = k$

Definiert man $a^* = \alpha_0 - i\alpha_1 - j\alpha_2 - k\alpha_3$, dann ist

$|a|^2 = a^*a = \alpha_0^2 + \alpha_1^2 + \alpha_2^2 + \alpha_3^2$ und

$a^{-1} = a^*/|a|^2$

Definieren Sie einen Datentyp für Quaternionen und eine Addition(+), Subtraktion(-), Multiplikation(*), Division(/).

12.4 Selbstdefinierte Zuweisungen

12.4.1 3D-Vektoren

Im Verzeichnis *F_loesungen* ist ein Programm *vec3.f90* mit einer unfertigen Algebra für 3D-Vektoren in kartesischen Koordinaten.

Ergänzen Sie die Operatoren +, -, ==, /= und .x. für das äußere Produkt (Kreuzprodukt), dazu noch die Funktion `spat(v1,v2,v3)`!

12.4.2 Bruchrechnung

Definieren Sie den Datentyp und die nötigen Operatoren, um folgende Formeln in Bruchrechnung berechnen zu können. Auch die Zuweisung zu einer reellen Variablen soll definiert und getestet werden.

```
  3+1/(7+1/16)

  3+1/(7+1/(15+1/(1+1/293)))

  2+1/3*(2+2/5*(2+3/7*(2+4/9*(2+5/11*(2+6/13*(2+7/15*(2+8/17* &
 (2+9/19*(2+10/21*(2+11/23*2)))))))))
```

Man muss beim ersten Beispiel nur den innersten Bruch so schreiben, dass er erkennbar vom Typ Bruch ist, also z.B. `b(1,16)`.
Wer nicht viel Zeit hat, macht alles nur für die beiden ersten Formeln.

12.4.3 Zeichenfolgen verschiedener Länge

Datenobjekte mit dem vordefinierten Typ Character haben eine feste Länge. Schreiben Sie ein Modul mit einer Definition eines Zeichentyps **string** mit variabler Länge, mit einer Maximallänge von 80, und auch die Unterprogramme zum

- Zuweisen einer Character-Variablen zu **string**
- Zuweisen eines **strings** zu einer Character-Varialen
- Rückgabe von `len` eines **strings**
- Verketten zweier **strings**

12.4.4 Langzahlen

Schreiben Sie einen Modul ***char_ari***, der die Addition beliebig langer Zeichenfolgen mit natürlichen Zahlen möglich macht. Man soll also schreiben können:

```
program lange_Zahlen
use char_ari
character(len=20) :: c
character(len=*),parameter :: &
                     a = '33333333333333'
character(len=7)           :: b = '555'
write(unit=*,fmt=*) '999999999999' + '1'
c = a + b
write(unit=*,fmt=*) c
```

Sind, wie am Beispiel ***b='555'***, am Ende Leerzeichen, so soll nur der Teil vor diesen Leerzeichen als Zahl interpretiert werden. Führende Leerzeichen werden als 0-Ziffern interpretiert.

Eine leere Zeichenfolge als Operand darf als 0 gedeutet werden.

Das Ergebnis soll in jedem Fall die Länge "1+ Maximum der beiden Operandenlängen"haben. Führende Nullen sind erlaubt.

Wenn in dem Zahlenteil der Zeichenfolgen etwas vorkommt, was keine Ziffer ist, dann soll das Ergebnis aus lauter '?' bestehen.

Kapitel 13

Pointer

13.1 Die Pointer-Zuweisung

13.1.1 Wie das Leben so spielt

Spiel des Lebens von Conway.
Programmieren Sie das Spiel zunächst auf der einfachsten ”Zuchtschalen”-Topologie! Eine Integer-Matrix S(ny,nx) geringster Genauigkeit wird zur Beschreibung eines zweidimensionalen Gitters verwendet, dessen Knotenpunkte (Zellen) entweder tot oder lebendig sein können.

Der Rand oben, unten, links und rechts bleibt bei dieser Topologie immer tot. Von den inneren Zellen werden zu Beginn einige ”geimpft”.

Am einfachsten geschieht das durch Einlesen. In S dürfen nur die Zahlen 0 für tot oder 1 für lebendig stehen! Eingabebeispiel: game_of_life.data .

Die inneren Elemente werden nun nach folgendem ”Naturgesetz” einige Male neu besetzt: Eine tote Zelle wird immer dann zum Leben erweckt, wenn sie genau drei lebende Nachbarn hat; eine lebende Zelle überlebt in der nächsten Generation nur dann, wenn sie zwei oder drei lebende Nachbarn hat- bei wenigeren stirbt sie an Vereinsamung, bei mehr wegen Übervölkerung. Als Nachbarn einer Zelle S(iy,ix) gelten alle 8 Zellen, bei denen ix und iy einzeln oder gemeinsam um 1 nach unten oder oben abweichen.

Beachten Sie bitte, dass die Lebensentscheidungen für alle Zellen gleichzeitig geschehen müssen. Versuchen Sie, die Lebensberechnung ohne `do`-Schleifen mit Teilfelder-Pointern zu lösen.

Natürlich soll das Anfangsmuster und das während jeder Iteration entstandene Muster ausgegeben werden. Wenn Sie ny und nx geschickt so wählen, dass die Zuchtschale gerade einen Bildschirm füllt, können Sie sozusagen einen Film drehen.

Versuchen Sie, wenn Sie noch Zeit haben, das Programm für die Topologie eines Zylinders (der linke Rand hat den rechten Rand als Fortsetzung - und umgekehrt) oder eines Möbius-Bandes (der linke Rand hat den rechten Rand als Fortsetzung, aber in umgekehrter Durchlaufsrichtung- und umgekehrt) oder eines Torus (beide Ränder grenzen in normaler Zuordnung aneinander) oder einer Kleinschen Flasche (oben-unten sind normal verbunden, rechts-links überkreuz) zu variieren!

Für jeden der Fälle kann man, wenn man es so organisieren will, ein eigenes Unterprogramm schreiben, das jeweils die Neuberechnung durchführt.

13.1.2 Seifenblasen 1

Schreiben und testen Sie ein Unterprogramm *premat(fxy)*; es soll die reelle Matrix *fxy* mit Werten vorbesetzen. Die Matrix ist im aufrufenden Programm mit *real fxy(0:n, 0:n)* vereinbart. N soll eingelesen werden. In dem nun folgenden Besetzungsmuster ist z.B. n=16 gewählt, nehmen Sie aber auch höchstens n=32. Setzen sie in der Matrix s=0.5, R=1.0, r=0. !

```
srrrrrrrrrrrrrrs
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
RssssssssssssssR
srrrrrrrrrrrrrrs
```

Versuchen Sie, diese Aufgabe mit Pointern auf Teilfelder zu lösen!

Der so gebildete Rand soll in der folgenden Aufgabe durch eine Seifenblase (Minimalfläche) verbunden werden.

13.1.3 Seifenblasen 2

Ein Unterprogramm *flamat(fxy)*! Es berechnet aus der vorbesetzten Matrix *fxy* (siehe vorige Aufgabe) iterativ eine neue Matrix.

Eine Iteration berechnet jedes innere Element von *fxy* neu als Mittelwert der vier Elemente, die links, rechts, über und unter dem betrachteten Element liegen.

Das Verfahren konvergiert bei größeren *n* nur langsam. Nehmen Sie also für die Übung höchstens n=32.

Die Rechnung soll hier in der Übung der Einfachheit halber ohne Konvergenzabfrage nach ca. 50 Iterationen beendet werden.

Versuchen Sie, alle Rechnungen mit Pointern auf Teilfelder zu programmieren (5 Pointer).

13.1.4 Seifenblasen 3

Zeichnen Sie die eben berechnete Matrix *fxy* als Funktionsfläche mit Höhenlinien (GR-Software)!

xx(0:n) und *yy(0:n)* sind im folgenden Programm reelle Felder mit den n+1 Stützpunkten auf der X- und Y-Achse (im Beispiel zwischen 0 und 1; mit gleichen Abständen **vorbesetzen** !).

Nw ist die gewünschte Anzahl der Höhenlinien. W(nw) ist ein reelles Feld mit den Höhenwerten, zu denen Linien erscheinen sollen; im vorliegenden Beispiel sollten Werte von 0 bis 1 mit gleichem Abstand **vorbesetzt** werden.

```
use grsoft_interface_block
integer :: ier
integer,parameter ::n=32,nb=n+1,&
   nw=21,lar=200000
real ::fxy(0:n,0:n),xx(0:n),yy(0:n),&
   w(nw),ar(lar)
call grstrt(35,8)
call gr3dim(lar,ier)
call gr3bks(1,5,3,4,5,6,7,8)
call gr3hhl(ar,ier,nb,1,xx,nb,1,yy,nw,&
  w,nb,fxy,0,1,2)
call gr3rot(ar,ier,'Z',30.,'X',-60.,&
   'Y',0.)
call gr3plo(ar,ier,'HID')
call grend
```

13.2 Zuweisung (=) von Pointern und Strukturen mit Pointern

13.2.1 Beliebig lange Zahlen

Erzeugen Sie einen Datentyp für beliebig lange Integer-Zahlen. Dazu soll die Addition, Subtraktion und wenigstens ein Vergleichsoperator geschaffen werden. Zu dieser Aufgabe wird ein Teil des Lösungsprogramms zur Verfügung gestellt.

13.3 Pointer-Funktionen, Pointer-Parameter

13.3.1 Nur auf das Positive hinweisen

Eine Funktion *p_pos* mit Pointer-auf-reellen-Vektor-Ergebnis soll von einem Pointer-Vektor *vin* abhängen. Der Ergebnisvektor soll in dem Unterprogramm mit der richtigen Länge angelegt werden. Er soll alle positiven Elemente von *vin* an die Aufrufstelle zurückgeben. Wenn es kein positives Element gibt, soll er nicht-zugeordnet sein.

Im aufrufenden Programm soll dieser Ergebnisvektor das aktuelle Argument ersetzen: vin => p_pos(vin). Sorgen Sie dafür, dass kein toter Speicherplatz zurückbleibt!

13.3.2 Primzahlen bis n

Eine Funktion *alle_prim(n)* soll alle Primzahlen bis n als Integer-Vektor-Pointer zurückgeben.

Hinweis zum Algorithmus: Die Anzahl der Primzahlen bis n (π (n)) ist kleiner als 1.26*n/ln(n). Man kann also die Primzahlen zunächst auf einem allozierbaren Vektor *pr* dieser Länge sammeln. pr = (/ 2, 3, 5 ... /) .

Es soll nicht das Sieb des Eratosthenes benutzt werden, sondern von 7 bis n abwechseln 4 und 2 addiert werden. Die Zahlen dieser Folge (*möpri*) sind schon nicht durch 2 oder 3 teilbar. Bei jeder dieser Zahlen *möpri* wird geprüft, ob sie durch eine der Zahlen von *pr* teilbar ist (pr(i), i=3 ..., bis

pr(i)**2 $>$ *möpri*). Wenn sich herausstellt, dass die Zahl nicht teilbar war, wird sie ans bisherige Ende von *pr* angefügt.

Die Fälle n$<$6 erfordern eine Sonderbehandlung.

13.4 Noch einige Beschränkungen bei Pointern

13.4.1 Sortieren von Individualisten

Es soll *n_texte_max* und dann maximal *n_texte_max* Texte eingelesen werden. Dabei kann man sie auch zählen: n Texte.

Man kann dazu einen Vektor mit n Elementen von einer Struktur anlegen, deren einzige Komponente ein Pointer auf ein `character`(len=1)-Feld ist. Diese Pointer sollen auf die eben eingelesenen Texte zeigen.

Nach diesen Vorarbeiten sollen die Texte sortiert ausgegeben werden.

13.5 Verkettete Listen

13.5.1 Vor-Zurück-Liste

Schreiben Sie ein Programm, das eine Datei mit ganzen Zahlen liest. In jeder Zeile wird nur das erste Wort als Zahl gelesen. Die Zahlen sollen im allgemeinen in derselben Reihenfolge wieder ausgegeben werden. Wenn aber eine ungerade Anzahl von negativen Zahlen gelesen wurde, sollen die Zahlen bis zur nächsten negativen Zahl (bzw. bis Dateiende) in umgekehrter Reihenfolge erscheinen.
Beispiel (hier stehen die Zahlen nebeneinander):

1,2,3,-4,5,4,2,3,-6,-6,4,23,-7,0,2,7
1,2,3,-4,3,2,4,5,-6,-6,23,4,-7,0,2,7

Wenn das erste Wort einer Zeile nicht als Zahl lesbar ist, darf das Programm abbrechen. Es sollte aber die zugehörige Zeilennummer ausgeben.

Am Ende der Datei soll ausgegeben werden, welche Satznummer der Endfile-Satz haben würde.
Die Aufgabe soll hier durch eine einfach verkettete Liste (Stack) gelöst werden.

13.5.2 Zettelkasten

Es soll eine unbekannte Zahl von Worten eingelesen werden, jedes Wort in einer Zeile. Nachdem alle Worte gelesen wurden, sollen sie wieder alphabetisch sortiert ausgegeben werden - und dazu die Anzahl, wie oft sie jeweils vorkamen.

- Realisieren Sie das mit einer einfach verketteten sortierten Liste! Ein Wort wird, sobald es gelesen wurde, an der richtigen Stelle in der Liste eingefügt. Am Ende soll die Liste gelöscht werden.
- Lösen Sie das Problem mit einem binären Suchbaum! Auch hier soll am Ende alles gelöscht werden. Hinweis : Nur mit rekursiven Unterprogrammen ist das Problem relativ einfach programmierbar.

13.5.3 last-in-first-out

Erzeugen Sie eine einfach verkettete Liste mit variabellangen `character`(1)-Vektoren als Werten. Die Zeichen-Vektoren sollen aus gleichlangen Zeichenketten gebildet werden, die man mit Hilfe des A-Formats einliest. Sie beginnen mit dem ersten Nicht-Leerzeichen und enden mit dem letzten. Die Ausgabe der Zeilen soll in umgekehrter Reihenfolge wie die Eingabe erfolgen (last-in-first-out). Die Texte selbst sollen nicht rückwärts erscheinen.

Eine solche Liste ist einfacher zu erzeugen als die first-in-first-out-Liste. Ein schön geeignetes Eingabe-Beispiel ist *F_loesungen/walther.input*, wenn es in der ursprünglichen und rückwärtigen Reihenfolge ausgegeben wird.

13.5.4 last-in-first-out-sets

Erzeugen Sie eine einfach verkettete Liste mit variabellangen Integer-Vektoren als Werten. Die Vektoren sollen von den einzelnen Zeilen einer Datei mit maximal 100 Werten pro Zeile gelesen werden; die Zeilen bestehen dabei aus höchstens 200 Zeichen. Die Ausgabe der Zeilen soll in umgekehrter Reihenfolge wie die Eingabe erfolgen (last-in-first-out). Die Zahlen innerhalb der Zeilen selbst sollen nicht rückwärts erscheinen.

Eine solche Liste ist einfacher zu erzeugen als die first-in-first-out-Liste.

Eine mögliche Lösung des Einleseproblems:

```
character(len=200),dimension(8) :: indat
integer,dimension(100) :: inza
inza = huge(1)
write(unit=indat(2:),fmt=*) inza
do
  read(unit=*,fmt="(a)",iostat=ios) &
     indat(1)
  if (ios  ...
  read(unit=indat,fmt=*) inza
  n = size(inza) - count(inza==huge(1))
```

13.5.5 Neuer Leithammel

Gegeben sei eine Liste aus Elementen von folgendem Typ:

```
type,public :: cell
    integer             :: inhalt
    type(cell), pointer :: next
end type cell
```

Die Variablen CURRENT und FIRST seien vom Typ CELL, FIRST zeige auf das erste Element einer zunächst gebildeten Liste.
Welche Anweisungen bewirken das Einsetzen von CURRENT (einem neuen Element) am Anfang der Liste?

13.5.6 Verkettete Individualisten

Lesen Sie von einer Datei Zeilen mit Text ein! Von dem Text soll alles, was zwischen dem ersten und letzten Nicht-Leerzeichen steht, als `character`(1) - Vektor in einer doppelt verketteten Liste abgespeichert werden. Wahrscheinlich ist dabei TRANSFER nützlich.

Am Ende sollen die Texte in unveränderter und in rückwärtiger Reihenfolge ausgegeben werden.

13.5.7 Sortierte Liste von Namen mit GermanUmlauts

Eine unbekannte Anzahl von Namen soll eingelesen und gleich in einer Liste an der richtigen Stelle abgelegt werden (alphabetisch sortiert). *Richtig* einsortiert ist ein Name, wenn in ihm und den Vergleichsnamen beim Vergleichen alle Umlaute durch die entsprechenden Doppelvokale ersetzt sind.

Am besten speichert man in jedem Listenelement den Namen und den Ersatznamen.

Am Ende soll die sortierte Liste ausgedruckt werden. Gleichzeitig sollen alle benutzten Speicherplätze der Liste wieder freigegeben werden.

Hinweis: Man kann auch `iso_varying_string` aus dem Anhang benutzen.

Kapitel 14

Vordefinierte Programme für Datum und Uhrzeit

14.1 system_clock

14.1.1 Verrinnende Zeit

Schreiben Sie eine Subroutine `idle(secs)`! Das reelle `Secs` ist der Bruchteil einer Sekunde. Wenn `Secs` 0.5 ist, soll `idle` möglichst genau eine halbe Sekunde aktiv sein. Wenn man mehr als eine Sekunde verlangt, soll trotzdem nur eine Sekunde Wartezeit entstehen.

Mit Hilfe dieser Routine und `advance="no"` soll eine Folge von Punkten ..... auf dem Bildschirm nebeneinander ausgegeben werden: alle `Secs` Sekunden ein neuer Punkt.

Anhang A

Zeichenfolgen variabler Länge

A.0.2 GermanUmlauts

Schreiben Sie eine Funktion vom Typ *varying_string*, die als Eingabeparameter einen normalen Characterstring hat. Falls in diesem String irgendwo ein Umlaut steht, soll er im Ergebnis in einen Doppelvokal umgewandelt werden: Ä in Ae, ä in ae usw., alle anderen Zeichen sollen übernommen werden.

Lesen Sie von einer Datei einige Namen mit Umlauten ein! Die Namen fangen mit Großbuchstaben (zu den Buchstabe zählen wir hier auch die Umlaute) an. Sie haben wenigstens die Länge drei.

Geben Sie die ursprünglichen Namen und die mit Hilfe unserer Funktion umgewandelten nebeneinander aus.

A.0.3 Mit dem Beil programmieren

Ein Programm soll ein Fortran-Programm mit freier Eingabeform einlesen und wieder ausgeben. Dabei soll jede Zeile, die mit einem Kommentar endet, in zwei Zeilen ausgespalten werden. Die zweite besteht nur aus dem Kommentar.

A.0.4 Längstes Abgeschriebenes (ggts)

Schreiben Sie ein Unterprogramm, das den längsten gemeinsamen Teilstring zweier modifizierter Zeichenfolgen A und B bestimmt und dem aufrufenden Programm übergibt. Die Modifikation soll darin bestehen, dass von A und B im Unterprogramm zunächst Kopien angefertigt werden, in denen alle Folgen von mehr als einem Leerzeichen durch ein einziges Leerzeichen ersetzt sind. Von diesen Kopien soll der längste gemeinsame Teilstring bestimmt und dem aufrufenden Programm zur Verfügung gestellt werden. (Vorsicht: Nicht ganz so einfach).

A.0.5 Flattertexte

Schreiben Sie ein Programm, das eine Nachricht von einer ursprünglich für Druckerausgabe bestimmten ASCII-Datei zeilenweise einliest und anschliessend die Zeilen formatiert wieder ausgibt. Die Datei enthält nur alphanumerische Zeichen und Leerzeichen.

Unter Formatieren sei folgendes verstanden:

- Von jeder Zeile sollen maximal m Zeichen eingelesen werden (m einlesen).
- Bei der Ausgabe soll der Inhalt jeder Zeile so angeordnet sein, dass das erste Wort in der ersten Spalte beginnt, und das letzte rechtsbündig in Spalte n endet (n einlesen). Wenn eine

Zeile nur ein Wort enthält, soll dies - falls möglich - linksbündig ausgegeben werden. Ein Wort ist natürlich eine Zeichenkette, die keine Leerzeichen enthält.

- falls erforderlich, sind bei der Ausgabe Leerzeichen zwischen den Worten möglichst gleichmäßig zu verteilen.


JÜLICH
FORSCHUNGSZENTRUM