Programmieren in Fortran90 / Fortran2003 (Auszug der wichtigsten Sprachelemente)

Besondere Zeichen

`!`	Kommentar (bis Zeilenende)
`;`	zwischen Befehlen in einer Zeile
`**`	Potenz: `a**2` $= a^2$
`"`	Zeichenkettenbegrenzung
`//`	Zeichenketten verbinden
`%`	Typkomponente: `A%x` $= A_x$
`:`	Indexbereich
`*`	freier Wert, z. B. `fmt=*`

Keine Unterscheidung zwischen Groß- und Kleinschreibung.

Hauptprogramm

```
program ⟨programmname⟩
  [use ⟨modulname1⟩,...]
  Deklarationen
  Hauptprogrammcode
end program
```

Module

```
module ⟨modulname⟩
  Deklarationen
[contains
  Unterprogramme, Funktionen]
end module
```

Hinweis: Module und das Hauptprogramm dürfen in einer Datei stehen.

Unterprogramme und Funktionen

```
subroutine ⟨name⟩(⟨par1⟩,⟨par2⟩,...)
      Deklaration der Parameter
  Deklaration lokaler Variablen
  Unterprogrammcode
end subroutine
```

Hinweise: Variablen werden stets als Referenz übergeben.

Aufruf von Unterprogrammen (interne, eigene und externe): `call ⟨name⟩(...)`.

```
function ⟨name⟩(⟨par1⟩,⟨par2⟩,...)
      Deklaration von ⟨name⟩
      Deklaration der Parameter
  Deklaration lokaler Variablen
  Unterprogrammcode
  ⟨name⟩=Rückgabewert
end function
```

Programmstrukturen

```
if (Bedingung) Befehl
```

```
if (Bedingung) then
  Danncode
[else
  Sonstcode]
end if
```

```
do ⟨var⟩=von,bis
  Schleifencode
end do
```

```
do while (Bedingung)
  Schleifencode
end do
```

Logische Ausdrücke

`==`, `/=`	gleich, ungleich
`<`, `<=`, `>`, `>=`	kleiner/größer(gleich)
`.not.`	nicht
`.and.`, `.or.`	und, oder

Variablen deklarieren

```
Typ,Attribute::⟨var1⟩,⟨var2⟩,...
```

```
Typ,Attribute::⟨var1⟩=Wert
```

Typen:

`integer`	Ganzzahl
`real`	Fließkommazahl
`logical`	`.true.` oder `.false.`
`character`	Zeichen(kette)
`type(⟨typ⟩)`	selbstdefinierter Typ

Attribute (optional):

`parameter`	Konstante
`dimension(Indizes)`	Datenfeld
`pointer`	dynam. Variable
`save`	statische Var.

Beispiele:

```
integer,dimension(4,3)::Gitter
real,parameter::pi=3.14159265359
character(len=*),parameter::WE="Euro"
```

Achtung: Initialisierung erzeugt implizietes `save`-Attribut. Daher Wertzuweisung nur für Konstanten (`parameter`) verwenden!

Neue Datentypen definieren

```
type ⟨name⟩
  Komponentendekl. (wie Variablen)
end type
```

Indexkonzept und der ":"

Felder beginnen normalerweise bei 1. `real,dimension(n)::x` deklariert deshalb `x(1)` bis `x(n)`. Dagegen startet man mit `dimension(0:n)` bei `x(0)`.

Der Doppelpunkt beschreibt einen Indexbereich. Man kann sogar die Grenzen weglassen, wenn man sie (noch) nicht kennt. `dimension(:,1:3)` ist also ein ($n\times3$)-Feld, n beliebig. Verwendung: bei Parametern von `subroutine` bzw. `function` oder bei dynamischen Feldern.

Beim Zugriff auf Feldvariablen kann man auch ":" verwenden oder die Indizes ganz weglassen, denn viele Operationen lassen sich auch mit ganzen Feldern ausführen. Bsp.: `x(1:10)=0` ⇔ `x(:)=0` ⇔ `x=0` ⇎ `x(10)=0`.

Dynamische Variablen und Felder

Verwendung von Variablen oder Komponenten mit `pointer`-Attribut:

```
real,dimension(:,:),pointer::a
allocate(a(5,2))
a(4,:)=7          (a(4,1)=7, a(4,2)=7)
deallocate(a)
```

Hinweis: Allozierte Pointer weden automatisch dereferenziert. (Für C-Programmierer: "`*`" und "`&`" einfach weglassen.) Zuweisung von Adressen mit "`=>`".

Für Felder ist statt `pointer` auch das `allocatable`-Attribut zulässig (uneingeschränkt erst ab Fortran2003), Vorteil: es wird ein echtes Feld erzeugt, anstatt eines allozierten Zeiger-Feldes.

Ein- und Ausgabe

```
print Format,⟨var1⟩,⟨var2⟩,...
```

```
read Format,⟨var1⟩,⟨var2⟩,...
```

```
write(Spezifikation) ⟨var1⟩,⟨var2⟩,...
```

```
read(Spezifikation) ⟨var1⟩,⟨var2⟩,...
```

Formate:

`*`	automatisches Format
`"(a)"`	für Typ character
`"(f)"`	für Typ real
`"(i)"`	für Typ integer
`"(l)"`	für Typ logical

Hinweis: Ggf. Anzahl davorschreiben. Mehrere Angaben durch Komma trennen. Breite und Nachkommastellen (nur `f`) anhängen. Bsp: `"(2a,i4,f7.2)"` = 2 Zeichenketten, ein Integer und ein Real (xxxx.xx).

Spezifikationen:

`unit=*`	STDIN/OUT
`unit=Nr`	Dateinummer (s.u.)
`fmt=...`	siehe Formate
`advance="no"`	ohne Zeilenumbruch

Hinweis: `advance` geht nur mit `fmt="(...)"`.

Dateien öffnen und schließen

```
open(unit=Nr,file=Dateiname,Modus)
```

```
close(unit=Nr)
```

Modus (mehrere Angaben erlaubt):

`status="old"`	Datei vorhanden
`status="replace"`	falls Datei ex., diese ersetzen
`action="read"`	zum Einlesen
`action="write"`	zum Ausgeben
`position="rewind"`	am Anfang starten
`position="append"`	am Ende anfügen

Interne Funktionen u. Unterprogramme

`abs`	Betrag
`exp`, `log`	e^x, $\ln(x)$
`sin`, ...	Trigonometrie (Bogenmaß)
`asin`, ...	Arcus-Funktionen (v. Bgmaß)
`min`, `max`	Minimum, Maximum
`mod(a,b)`	a modulo b
`int`	Ganzzahl (abschneiden)
`sqrt`	$\sqrt{x}$
`trim`	Leerzeichen abschneiden

Unterprogramme (mit `call`):

`cpu_time(x)`	Rechenzeit (Sek.)
`random_number(x)`	Zufallszahl $x \in [0,1[$
`random_seed()`	Zufallsgenerator neu initialisieren

Fortran Compiler (für Linux, Win, etc.)

G95 (www.g95.org)

Aufruf: `g95` *Optionen [-o Ausgabedatei]*

Optionen:

`-r8`	doppelte Genauigkeit (real = "double")	`-fimplicit-none`	Fehlermeldung bei undeklarierten Variablen
`-O3`	maximal optimieren (kompiliert langsamer)	`-std=f2003`	genau auf Fortran2003-Standard prüfen
`-march=...`	optimieren für spezielles System, siehe gcc (z.B. pentium{2,3,4,-m}, athlon{-xp,64}	`-fone-error`	Kompilieren beim ersten Fehler abbrechen
		`-fbounds-check`	Indexbereich von Feldern prüfen (langsamer)
		`-ftrace=full`	detail. Fehlermeldung bei Absturz (langsamer)

Gfortran (ab GCC-4.0 in der GNU-Complier-Serie enthalten)

Aufruf: `gfortran` *Optionen [-o Ausgabedatei]*

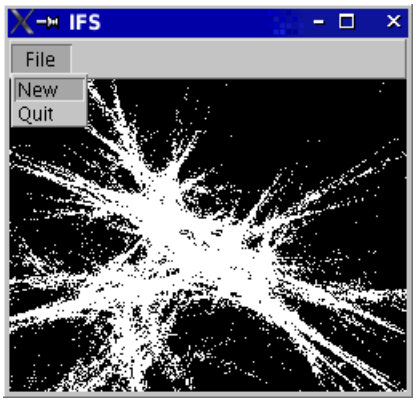
Optionen: siehe g95, jedoch `-fdefault-real-8` statt `-r8`

Grafische Oberfächen mit Japi (www.japi.de)

JAPI: plattformunabhängiges, programmiersprachenübergreifendes (bisher: Basic,C,Pascal,Fortran,Ada,Euporia) Java-API-Interface.
1.) JAPI-Bibliothek (libjapi.a) downloaden oder Quellen downloaden und mit make übersetzen (erfordert gcc).
2.) Fortran-Interface (japi.f90) downloaden (siehe Rubrik Fortran/Imagine1)
3.) beide o. g. Dateien (ggf. mit Pfad) zusätzlich dem Fortran-Compiler übergeben

Beispielprogramm mit Japi-Oberfläche (ifs.f90)

Compile: `gfortran ifs.f90 japi/japi.f90 japi/source/libjapi.a -fimplicit-none -fdefault-real-8 -O3 -o ifs.out`




```
module ifs_module

  integer::numberofmaps
  type lin_map
    real,dimension(2,2)::m
    real,dimension(2)::s
  end type
  type(lin_map),dimension(10)::maps

  public::ifs_init,ifs_iter

contains

subroutine ifs_init
real::r
integer::j
  call random_number(r)
  numberofmaps=int(2+2*r)
  do j=1,numberofmaps
    call random_number(maps(j)%m)
    maps(j)%m=1-2*maps(j)%m
    call random_number(maps(j)%s)
  end do
end subroutine

function ifs_iter(p) result (q)
real,dimension(2)::p,q
real::r
integer::j
  call random_number(r)
  j=1+int(r*numberofmaps)
  q=matmul(maps(j)%m,p)+maps(j)%s
end function

end module


program ifs
  use japi
  use ifs_module

  integer::frame,menubar,cv,obj
  integer::m_file,m_new,m_quit
  integer::width,height,x,y,it
  real,dimension(2)::point
  logical::finish

  ! initialize japi, frame, menus
  if(.not. j_start()) call j_quit()
  width=320; height=240
  frame=j_frame("IFS")
  call j_setborderlayout(frame)
  menubar=j_menubar(frame)
  m_file =j_menu(menubar,"File")
  m_new  =j_menuitem(m_file,"New")
  m_quit =j_menuitem(m_file,"Quit")
  cv=j_canvas(frame,width,height)
  call j_pack(frame)
  call j_show(frame)
  ! initialize ifs programm
  call ifs_init; it=-1000; point=0
  call j_setcolor(cv,255,255,255)
  call j_setcolorbg(cv,0,0,0)
  finish=.false.
  do while(.not. finish)
    obj=j_getaction()
    ! menu events
    if(obj==frame) finish=.true.
    if(obj==m_quit) finish=.true.
    if(obj==m_new) then
      call j_setcolorbg(cv,0,0,0)
      call ifs_init; it=-1000; p=0
    end if
    ! frame/canvas resize
    if(obj==cv) then
      width=j_getwidth(cv)
      height=j_getheight(cv)
      call j_setcolorbg(cv,0,0,0)
    end if
    ! iteration
    point=ifs_iter(point); it=it+1
    if (it>0) then
      x=(0.4+point(1)*0.2)*width
      y=(0.4+point(2)*0.2)*height
      call j_drawpixel(cv,x,y)
    end if
  end do
  call j_quit()
end program
```