Simulation course FYSM350

Fortran : Vesa Apaja email: vesa.apaja@gmail.com
MD : Hannu Häkkinen
MC : Juha Merikoski

www info:

Course homepage:

http://www.phys.jyu.fi/homepages/merikosk/Simu2006.html

(a link to this page is in korppi)

Useful material:

http://www.csc.fi/oppaat/f95/

Introduction

- History

  "Dad's FORTRAN"

  First version born late 50's at IBM

  First standard 1966 : FORTRAN 66

  Second standard 1977: FORTRAN 77

  ---------- prehistory ----------

  Third standard 1990 : Fortran 90

  Fourth standard 1995 : Fortran 95

  Fifth standard 2003 : Fortran 2003

Describing his early work on FORTRAN, John Backus said:-
We did not know what we wanted and how to do it. It just sort of grew. The first struggle was over what the language would look like. Then how to parse expressions - it was a big problem and what we did looks astonishingly clumsy now....

Free compilers

Intel Fortran (ifort or ifc)

http://www.intel.com/software/products/compilers/flin/noncom.htm

- Windows or linux, any Intel or AMD processor
- Optimising f95 compiler with 2003 “readiness”
- Free for non--commercial use; Windows version is only evaluation copy
- After registering you get a license number
- Easy to install (linux: less than 5 mins)

Gnu g95

http://g95.sourceforge.net/

Source code available - written in C :^)

precompiled binaries for

Linux x86, Cygwin x86, Windows x86, Powerpc, FreeBSD x86,
Sparc Solaris, Linux IA64, Linux x86_64/EMT64, Linux Alpha,
Irix Mips *etc.*

Commercial compilers

Often linked to commands **f90** or **f95**

- Pathscale **pathf90**
- PGI (Portland group) **pgf90**
- MIPSPRO (SGI Irix machines)
- Lahey (Fujitsu) **lf95**
- Absoft
- Compaq Fortran (Digital Fortran)

Intel: Compiler options

Try **ifort -help** or **man ifort**

Development phase:

ifort -check all program.f90

Production phase:

Always try : **-fast**

ifort -O3 program.f90 Use always at least this

ifort -tpp7 -xP pentium 4 processor

Expect output lines "*remark: LOOP WAS VECTORIZED*"

g95: Compiler options

Development phase:

g95 -Wall -std=f95 program.f90

Production phase:

g95 -O3 -std=f95 program.f90

Try also : **-O2 , -funroll-all-loops -mtune=pentium4**

(practically all options of the gcc compiler apply)

Endianness: a binary data issue

This is too much for beginners, just skip it unless you already know what endianness means.

Linux machines are usually configured to be *little endian*, but many other Unix machines are *big endian*. So how to transfer binary data files among different machines?

Intel: compile programs using

ifort -convert big_endian program.f90

g95 : **g95 -fendian=big program.f90**

or

set the environment variable G95_ENDIAN

in bash shell : **export G95_ENDIAN=big**

in tcsh or csh : **setenv G95_ENDIAN big** **(or: BIG)**

then as usual **g95 program.f90**

Program 1

```
program test
  implicit none
  integer:: i
  real (kind(1.d0)) :: x,y
  complex (kind(1.d0)) :: c
  x = 5.d0
  y = 20.d0
  c = cmplx(x,y,kind(1.d0))
  print*,x,y,c
end program test
```

Good habit: use this always if your program is meant for serious use

Unused variable: waste of space

Double precision:
~ 16 decimal accuracy

Internal function "**cmplx**"

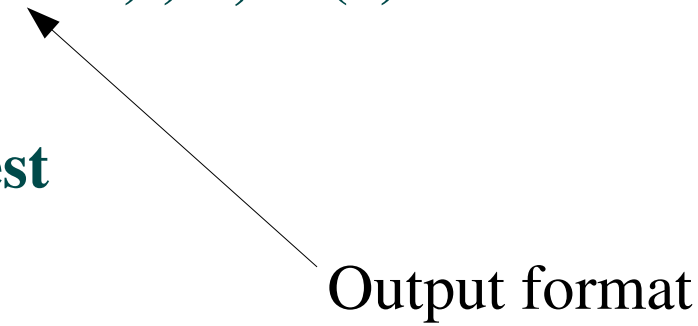
Convert two double precision numbers to a complex number

output:

```
 5.00000000000000       20.0000000000000
(5.00000000000000,20.0000000000000)
```

Program 2: formatted output on screen

```
program test
  implicit none
  integer, parameter:: dbl=kind(1.d0)
  integer:: i
  real (dbl), parameter:: dx=0.1d0
  real (dbl) :: x
  write(*,'(2a15)') 'x','sin(x)'
  do i = 1, 10
    x = (i-1)*dx
    write(*,'(2f15.10)') x,sin(x)
  end do
end program test
```



x	sin(x)
0.0000000000	0.0000000000
0.1000000000	0.0998334166
0.2000000000	0.1986693308
0.3000000000	0.2955202067
0.4000000000	0.3894183423
0.5000000000	0.4794255386
0.6000000000	0.5646424734
0.7000000000	0.6442176872
0.8000000000	0.7173560909
0.9000000000	0.7833269096

Formatted output

Most important format codes: examples

2f15.5 two floating point numbers, use 15 characters and give 5 decimals

f0.10 one floating point numbers, 10 decimals, as much field as it takes

a20 one 20 characters long field

d15.10 one number of the exponent type **10.d12** **5x** 5 spaces

Bad format codes may

- cause the program to terminate
- clobber the output to a useless form:

write(*,'(3f0.3)') 0.1234d0,0.5678d0,0.7890d0 0.1230.5680.789

Examples:

write(*,'(d15.5)') 1.123456789d12 0.11235D+13

write(*,'(g15.5)') 1.123456789d12 0.11235E+13

write(*,'(f15.5)') 1.123456789d12 ********** ← Field is too short

write(*,'(e15.5)') 1.123456789d12 0.11235E+13

Program 3: simple formatted output to a file

```
program test
   implicit none
   integer, parameter:: dbl=kind(1.d0)
   integer:: i
   real (dbl), parameter:: dx=0.1d0
   real (dbl) :: x
   write(12,'(2a15)') 'x','sin(x)'
   do i = 1, 10
      x = (i-1)*dx
      write(12,'(2f15.10)') x,sin(x)
   end do
end program test
```

"Automatic" file I/O:
Output to file fort.12

Unit number

Unit * : default
Unit 5 : keyboard
Unit 6 : screen

Formatted output cont'd

- Format strings in write statements:

write(55,”result = ”,f15.10)') res

- For frequently occurring formats use a separate **format** statement:

```
...
kin_E = 12.66666666666d0
pot_E = -1.44444444444d0
write(*,900) '   kinetic energy',kin_E
write(*,900) 'potential energy',pot_E
...
900 format(1x,a20," = ",f15.10)
...
```

```
   kinetic energy =   12.6666666667
potential energy =   -1.4444444444
```

Basic loop structures

```
j = 6
do i = 1, 10
   print*,i
   if(i==j) exit
end do
```

```
do i = 1, 10
   if(i<5) cycle
   print*,i
end do
```

```
i = 0
do while (i<3)
   i = i +1
end do
```

very useful !

e.g. **In force calculations particles don't exert force on themself, so for i:th particle you need to skip particle j if j=i**

If something goes wrong use **stop**:

```
subroutine mysub(x)
...
if(x<0.d0) stop 'mysub: negative x'
...
```

In bigger programs it's a good idea to let statements tell where they are ...

... and what exactly went wrong

Naming loops

```
iloop: do i = 1, 3
    jloop: do j = 1, 3
       kloop: do k = 1,3
          print*,i,j,k
          if(i==k) exit jloop
       end do kloop
    end do jloop
end do iloop
```

i	j	k
1	1	1
2	1	1
2	1	2
3	1	1
3	1	2
3	1	3

exit jloop

Program can **jump out of several nested loops**,
not just the present loop (**if(i==k) exit** would do that)

Equally important : In a long program it's hard to tell what loop does a bare end do really end.