Programming and Modelling (week 37)

C. Thieulot

Institute of Earth Sciences

September 2017

Instructions

Common mistakes

do loops

```
integer :: i,n
do i=1,n
   ...
end do
```

good habit

Formatting the source code makes a difference ...

```
program xample
integer i
real x

i=123
write(6,*) i
write(6,'(i1)') i
write(6,'(i4)') i
write(6,'(i16)') i

x=3.14159
write(6,*) x
write(6,'(f4.2)') x
write(6,'(f10.2)') x
write(6,'(f5.4)') x
write(6,'(f10.6)') x
write(6,'(es12.5)') x

write(6,'(a,i4,a,f5.2)') 'i=',i,', x=',x
end program
```

terminology

Symbol	Description	Symbol	Description
	space	=	equal
+	plus	-	minus
*	asterisk	/	slash
(	left paren	)	right paren
,	comma	.	period
'	single quote	"	double quote
:	colon	;	semicolon
!	shriek	&	ampersand
%	percent	<	less than
>	greater than	$	dollar
?	question mark		

write

```
write(*,*) 'And she's buying a stairway to heaven'
write(6,*) 'And she's buying a stairway to heaven'
```

write

```
write(*,*) 'And she's buying a stairway to heaven'
write(6,*) 'And she's buying a stairway to heaven'
```

$\Rightarrow$ do the same thing

write

```
write(*,*) 'And she's buying a stairway to heaven'
write(6,*) 'And she's buying a stairway to heaven'
```

$\Rightarrow$ do the same thing

```
write(1937,*) 'Another one bites the dust',i,sqrt(x)
```

write

```
write(*,*) 'And she's buying a stairway to heaven'
write(6,*) 'And she's buying a stairway to heaven'
```

$\Rightarrow$ do the same thing

```
write(1937,*) 'Another one bites the dust',i,sqrt(x)
```

$\Rightarrow$ writes text and numbers in file associated to unit 1937

formatting

```
program xample
integer i
real x

i=123
write(*,*) i
write(*,'(i1)') i
write(*,'(i4)') i
write(*,'(i16)') i

x=3.14159
write(*,*) x
write(*,'(f4.2)') x
write(*,'(f10.2)') x
write(*,'(f5.4)') x
write(*,'(f10.6)') x
write(*,'(es12.5)') x

write(*,'(a,i4,a,f5.2)') 'i=',i,', x=',x
end program
```

```
           123
*
 123
                123
   3.14159012
3.14
      3.14
*****
  3.141590
 3.14159E+00
i= 123, x= 3.14
```

fun facts (1)

Computers evaluate the right-hand side of an equation and put the result in the left hand side:

- `x=y+1` means that `x` receives the value `y+1`
- Careful: `x=x+1` means that `x+1` is first computed and its value stored in the variable `x` (thereby replacing the old previous value)

fun facts (2)

- Some languages distinguish between upper case and lower case.

fun facts (2)

- Some languages distinguish between upper case and lower case.
- Fortran does not. All these lines are equivalent:

```
integer imax
INTEGER IMAX
INTEGER imax
integer IMAX
InTeGeR ImaX
...
```

the executable

Fortran is a programming language (not a software)

the executable

Fortran is a programming language (not a software)

gfortran is the compiler which translates the fortran code you wrote into a binary code.

the executable

Fortran is a programming language (not a software)

gfortran is the compiler which translates the fortran code you wrote into a binary code.

If you compile a fortran file *myprogram.f90* as follows:
`> gfortran myprogram.f90`
the compiler generates an executable, which default name is *a.out*

the executable

Fortran is a programming language (not a software)

gfortran is the compiler which translates the fortran code you wrote into a binary code.

If you compile a fortran file *myprogram.f90* as follows:
`> gfortran myprogram.f90`
the compiler generates an executable, which default name is *a.out*

You can name the executable as follows:
`> gfortran myprogram.f90 -o myprogramexec`

the executable

Fortran is a programming language (not a software)

gfortran is the compiler which translates the fortran code you wrote into a binary code.

If you compile a fortran file *myprogram.f90* as follows:
`> gfortran myprogram.f90`
the compiler generates an executable, which default name is *a.out*

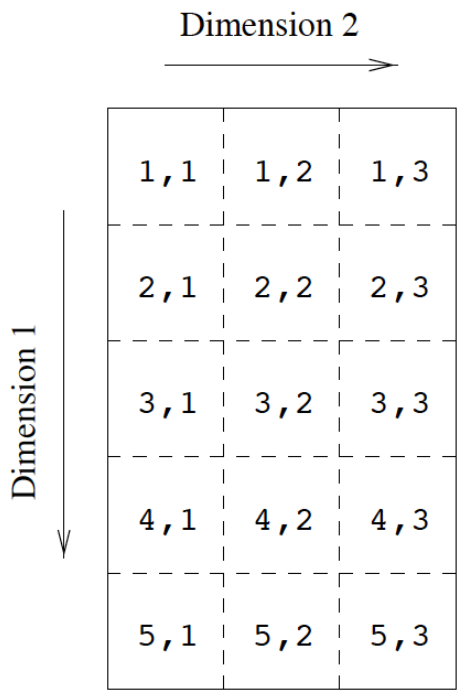
You can name the executable as follows:
`> gfortran myprogram.f90 -o myprogramexec`

To run the program, you then type:
`> ./myprogramexec`

arrays

```
integer tableau(15)
```

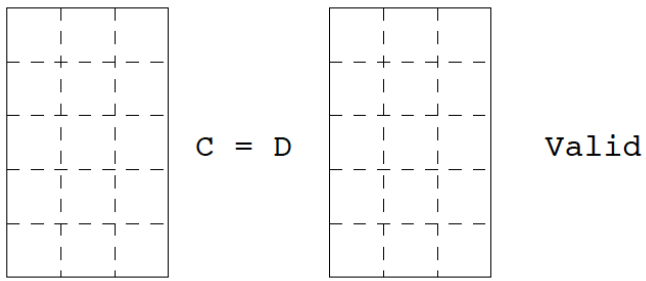

```
integer tableau(5,3)
```



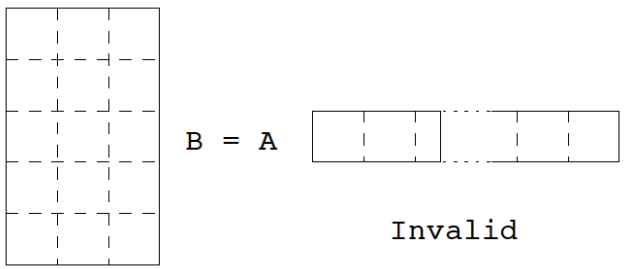
array conformance

```
real, dimension(15) ::  A
real, dimension(5,3) ::  B
real, dimension(5,3) ::  C
real, dimension(5,3) ::  D
```

- C=D is valid:



- B=A is not valid:



Integer arithmetics (1)

Let us start with:

```
program division
integer :: i,j
real :: x,y
i=1
j=2
write(6,*) 'i  =',i
write(6,*) 'j  =',j
write(6,*) 'i/j=',i/j
x=1.
y=2.
write(6,*) 'x  =',x
write(6,*) 'y  =',y
write(6,*) 'x/y=',x/y
end program
```

Integer arithmetics (1)

Let us start with:

```
program division
integer :: i,j
real :: x,y
i=1
j=2
write(6,*) 'i  =',i
write(6,*) 'j  =',j
write(6,*) 'i/j=',i/j
x=1.
y=2.
write(6,*) 'x  =',x
write(6,*) 'y  =',y
write(6,*) 'x/y=',x/y
end program
```

```
thebeast:progmod geogarfield$ ./a.out
 i   =           1
 j   =           2
 i/j =           0
 x   =   1.0000000
 y   =   2.0000000
 x/y =  0.50000000
```

Integer arithmetics (2)

```
program conversion
implicit none

integer :: i

i=7

write(6,*) i*1

write(6,*) i*1.

end program
```

Integer arithmetics (2)

```
program conversion
implicit none

integer :: i

i=7

write(6,*) i*1

write(6,*) i*1.

end program
```

```
thebeast:progmod geogarfield$ ./a.out
           7
   7.0000000
```

Integer arithmetics (2)

$$(-1)^n \frac{n(n+1)}{2n+1} \sqrt{2n^2 - n + 7}$$

translates into

```
integer ::  n
real ::  x
x=(-1.)**n * n*(n+1.)/(2.*n+1.)*sqrt(2.*n**2-n+7.)
```

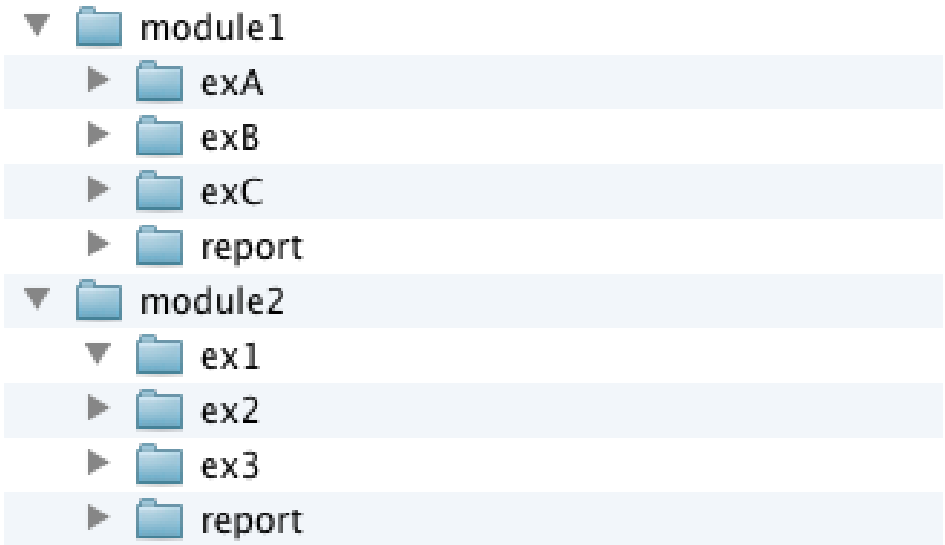
Mathematical operations

- addition `+`
- substraction `-`
- multiplication `*`
- division `/`
- exponentiation `**`
- trigonometric functions:
  `cos,sin,tan,acos,asin,atan,...`
- other functions: `sqrt,exp,log,log10,...`

shell commands

- *mkdir* (make directory)
- *ls -l* (list and display files in columns)
- *rm* (remove)
- *mv* (move a file, or rename it)
- *pwd* (print working directory)
- *more* (opens the file one screen at a time)
- *cp* (copy)

file structure



Example:

Use a do-loop construction to compute $n!$

Example:

Use a do-loop construction to compute $n!$

$1! = 1$
$2! = 1 \times 2 = 2$
$3! = 1 \times 2 \times 3 = 6$
$4! = 1 \times 2 \times 3 \times 4 = 24$
$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$
etc ...

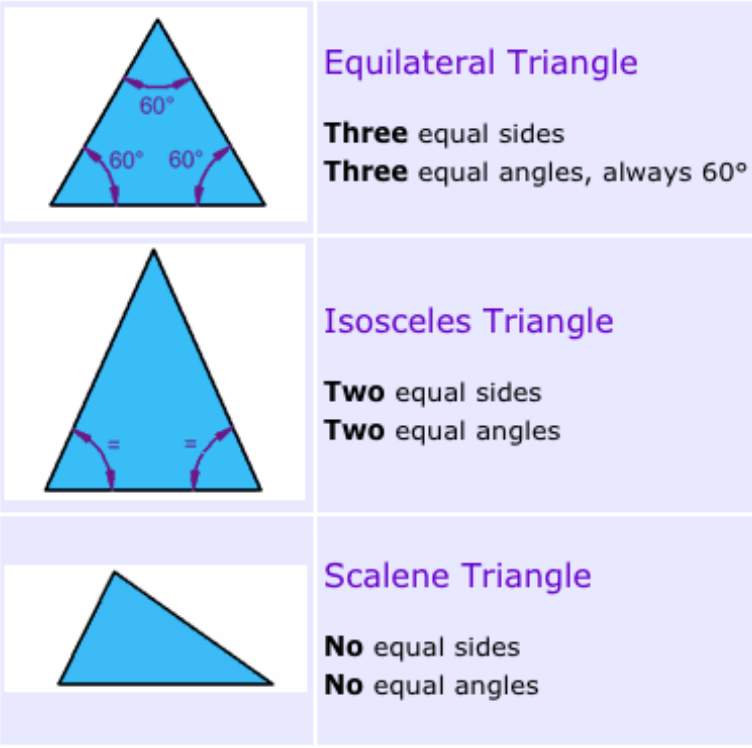
```
program factorial
implicit none

integer :: fact
integer :: i,n

write(6,*) 'enter a number'
read(5,*) n

fact=1
do i=1,n
   fact=fact*i
   write(6,*) i,'! =',fact
end do

end program
```

```
program factorial
implicit none

integer :: fact
integer :: i,n

write(6,*) 'enter a number'
read(5,*) n

fact=1
do i=1,n
   fact=fact*i
   write(6,*) i,'! =',fact
end do

end program
```

```
thebeast:progmod geogarfield$ ./a.out
           1 ! =           1
           2 ! =           2
           3 ! =           6
           4 ! =          24
           5 ! =         120
           6 ! =         720
           7 ! =        5040
           8 ! =       40320
           9 ! =      362880
          10 ! =     3628800
```

triangles & Co.

Write a program to accept the coordinates of three points and report back whether this points define an equilateral, isosceles, or scalene triangle.



(do three points always define a triangle ?)

Mathematical 'magic'

If you take a positive integer, halve it if it is even of triple it and add one if it is odd, and repeat, then you will ultimately obtain one.

Write a program to illustrate this.