Aide-memoire Fortran

Jean-Guy Caputo

1 Unités de programme

Principal

```
1234567
      program mm
      implicit none
      integer n,i
      real y
      double precision x
     & ,z
       ...
```

sous-programme

```
1234567
      subroutine trap(n,a,b,f,s)
      implicit none
      integer n,i
      real y
       ...
      end

      call trap(n,a,b,f,s)
```

fonction

```
1234567
      real function sinx(x)
      implicit none
      real x

      sinx = sin(x)/x
      return
      end
```

2 tests

```
if (condition 1)  then
```

```
    ....
elseif (condition 2) then
    ....
endif
```

3 boucles

boucle déterministe

```
do i=ideb,ifin,ipas
   .....
enddo
```

boucle indeterministe (do while, while..)

```
do
   ....
   if (condition) then
      ...
      exit
   endif
   ...
enddo
```

4 Tableaux

dynamique

```
real, dimension (:), allocatable :: a
```

pour allouer

```
allocate (a(n))
```

La **méthode de travail** est

1. on lit la dimension
2. on alloue le tableau
3. on remplit le tableau
4. on travaille
5. on desalloue deallocate (a)

Regle de travail

On alloue dans le main, pas dans les sous-programmes Les tableaux de travail sont passes une fois alloues dans le main. Ceci rend les sous-programmes portables, indépendants de la dimension.

Tableau statique

```
integer, parameter :: n=5
real a(n)
```

Ordre de stockage des tableaux en mémoire a(n,n)

$a_{11}a_{21}a_{31}\ldots$

l'indice le plus a gauche varie d'abord, en C c'est l'inverse

Opérations sur les tableaux conformants (meme profils)

N'importe quelle opération est effectuée terme a terme ainsi

```
a = b+ c*d
```

tout scalaire est conformant a n'importe quel tableau

```
a=0
```

section de tableaux

renversement de l'ordre d'un tableau

```
real a(20)
a(:) = a(20:1:-1)
```

Lecture de tableaux

```
real a(n,n)
read(*,*) a
```

ou

```
read(*,*) ((a(i,j),i=1,n),j=1,n)
```

lecture de a sur une seule ligne, convient pour $n < 5$ au delà lire ligne par ligne

```
do j=1,n
    read(*,*) (a(i,j),i=1,n)
enddo
```

5 Passage d'arguments entre programmes

5.1 arguments de sous-programmes

fortran passe les adresses des variables, pour un tableau c'est l'adresse du premier element qui est passée. Il faut donc préciser dans le sous-programme combien d'elements du tableau seront utilises.

Ex dans le main

```
program main
...
real a(n,n)
call sub (n,a)
...
end

subroutine sub(n,a)
real a(n,n)
...
end
```

5.2 zone commune (common) ou module

évite le passage par argument qui "pollue" l'appel. Ex : utilisation d'un solveur ode, celui-ci ne doit pas s'occuper des paramètres de l'équation différentielle

common

```
program main
...
common / texte/ x,y,z
...
end

subroutine sub(n,a)
real a(n,n)
common / texte/ x,y,z
...
end
```

texte est une chaine de caractères utilisée pour identifier le common. L'ordre des variables est important pour l'optimisation: on place d'abord les variables dans l'ordre de taille décroissant double précision/ réelles/ entiers

Dans le cas de plusieurs common (bloc common) je recommande d'utiliser un fichier "include". Ce fichier est inclus lors de la compilation.

```
program toto
...
```

```
include 'toto.inc'
...
end
```

fichier toto.inc (texte, pas une source fortran)

```
integer,parameter :: n=5
real a(n,n)
real x,y,z
integer i,j
common / texte1/ a
common / texte2/ x,y,z,i,j
```

module

Même fonction que le fichier include, sauf qu'il s'agit maintenant d'une unité de programme a part entière. Pour avoir accès aux variables du module mod_toto, une unité de programme doit comporter l'instruction "use mod_toto" en 2nde ligne

```
program toto
use mod_toto
...
end
```

fichier mod_toto.f

```
begin module mod_toto
integer,parameter :: n=5
real,save:: a(n,n)
real,save:: x,y,z
integer,save:: i,j
end module mod_toto
```

Le mot clé save permet de rendre les variables statiques afin qu'elles gardent la même valeur d'un sous programme a un autre

compilation plus délicate

```
# fait .mod et .o
plorenz.mod: plorenz.f
        $(FC) $(OPT) plorenz.f -c

# utilise plorenz.mod
mel.o: mel.f plorenz.mod
        $(FC) $(OPT) mel.f  -c

# executable
elor:  mel.o lor.o euler.o plorenz.o
        $(FC) $(OPT) mel.o lor.o euler.o plorenz.o  -o elor
```