Calculs scientifiques L3 : Introduction à Fortran

Historique : 1ère version de "FORmula TRANslator" née en 1954 chez IBM. Différentes versions/améliorations successives (Fortran77, Fortran90, Fortran95, Fortran2003, Fortran2008 ...) dont les plus utilisées sont Fortran77 et Fortran90.

1 Organisation des programmes

Pas de différences entre les majuscules et minuscules. Attention, éviter les accents !

En fortran77, 72 colonnes réparties comme suit :
- 1-5 : étiquettes
- 6 : annonce commentaire par le charactère 'c'
- 7-72 : instructions

En fortran90, le format est libre :
- le code peut commencer à la première colonne
- longueur maximale d'une ligne : 132 caractères
- commentaires introduits aussi par '!' à n'importe quelle colonne
- les instructions peuvent continuer à la ligne suivante si celle-ci commence par '&'
- ';' permet de séparer deux instructions écrites sur une même ligne

2 Structure

```
program main
Déclaration variables
Initialisation des paramètres
Commandes / calculs
end
```

3 Types de variables

Entiers : `integer` i, ou `integer`*2 i (simple précision) ; `integer`*4 i (double précision)

Réels : `real` a, ou `real`*4 a (simple précision) ; `real`*8 a (double précision)

Complexes : `complex` z, ou `complex`*8 z (simple précision) ; `complex`*16 z (double précision)

Chaines de caractères : `character` nom*n

N.B.: n peu être le nombre de charactères de la chaîne (ex. : `character` nom*4 pour nom=toto), ou bien n=1 si la chaîne n'est pas spécifiée.

Le nombre après '*' représente le nombre de byte (octets) occupés en mémoire par une variable. Ainsi, un réel à double précision occupe 8B, alors qu'un complexe occupe 16B. Un charactère occupe, quant à lui, 1B, voilà pourquoi il faut spécifier la longuer de la chaîne de charactères.

Des variables "spéciales" :

Vecteurs : `real`*8 t_r(n) (vecteur de n éléments réels à double précision), ou `integer`*4 t_i(n) (vecteur de n entiers)

Matrices : `complex`*16 m_c(m,n) (matrice de $m \times n$ éléments complexes)

Paramètres : ce sont des variables (d'un des types mentionnés) qui peuvent être partagées entre subroutines et fonctions (voir après). Elles doivent être déclarées, selon leur type, dans toute subroutine/fonction les utilisant, et doivent être initialisées lors de leur déclaration en tant que paramètre. N.B.: Leur valeur ne peut pas être changée.
Ex. : `parameter` (PI=3.141593, PI2=2*PI)

Common : ce sont aussi des variables (d'un des types mentionnés) qui peuvent être partagées entre subroutines et fonctions. Leur valeur, cependant, peut être changée par toute subroutine/fonction les utilisant.
Ex. : `common` /com/a,b

N.B.: Implicitement, les variables débutant par les lettre i, j, k, l, m et n sont de type entier et les autres sont de type réels, tous de simple précision. Pour éviter toute erreur, il vaut mieux annuler ces déclarations implicites par la ligne

```
implicit none
```

au début du code.

4 Opérateurs

Affectations

```
integer i,j
character*4 nom
i=2
j=5
nom='chat'      ! attention vérifier que nom a bien 4 lettres ...
```

Permutations

```
temp=a
a=b
b=temp
```

Opérateurs arithmétiques

+, -, *, / et ** pour la puissance

Opérations sur vecteurs et matrices

Depuis Fortran90 il est possible d'effectuer des opérations directement sur les vecteurs et sur parties de matrices (comme en Matlab).
Ex. : Si en Fortran77 il fallait implémenter une boucle pour sommer deux vecteurs A et B, en Fortran90 il suffit de faire C = A+B.
Les opérations sur matrices ne sont pas gérées directement. En revanche, on peut opérer sur des colonnes, ou des lignes, d'une matrice, ce qui correspond à opérer sur des vecteur.
Ex. : C = A(:,1)*B(:,2) génère un vecteur C qui est le produit élément-par-élément de la première colonne de la matrice 1 et de la deuxième colonne de la matrice B.

Casting

Il faut faire attention aux types des variables lorsqu'on effectue une opération entre elles. Si deux variables sont du même type le résultat de l'opération est lui aussi du même type.
Ex. : 3/5 donne 0, donc un `integer`.
Si les types ne sont pas les mêmes, le résultat prend le type "supérieur".
Ex. : 3.2/5 donne 0.64, c'est-à-dire un `real`.
On peut cependant forcer le type du résultat par un *casting* des opérands.
Ex. : `float`(3)/5 ou 3/`float`(5) donnent le même résultat que 3.0/5 ou 3/5.0.
Faire très attention à ce que le résultat d'une opération soit du même type que la variable à laquelle il va être affecté.
Ex. : a = 3.0/5 donne a=0 si a est un `integer`, alors que a=0.0 si a avait été défini comme un `real` (nonobstant le casting). Pour que le résultat soit correct, il faut donc que 1) a soit un `real` et que 2) il y ait un casting des opérands du type 3.0/5 ou `float`(3)/5

5 Lecture / écriture

Pour écrire : WRITE(*,*) 'Entrer la valeur de a'
Pour lire : READ(*,*) a

```
program permutation
implicit none
integer*4 a,b,temp
write(*,*) 'entrer la valeur de a'
read(*,*) a
write(*,*) 'entrer la valeur de b'
read(*,*) b
temp=a
a=b
b=temp
write(*,*) 'La nouvelle valeur de a est :', a,
&           'La nouvelle valeur de b est :', b
end
```

N.B.: Dans (*,*) le premier paramètre représente le canal de sortie pour l'écriture, le second indique le format d'écriture.
Dans ce cas, *,* veut dire écriture à l'écran et format libre. Autrement, il est par exemple possible d'écrire dans un fichier avec un `write`(1,*), ou 1 est le numéro associé à un tel fichier (voir après).

6 Boucles

6.1 Boucle conditionnelle IF

- IF / ELSE / END

```
if (condition) then
    instructions1
else
    instructions2
endif
```

- IF / ELSEIF / END

```
if (condition1) then
    instructions1
elseif (condition2) then
    instructions2
...
endif
```

Identificateurs logiques : `.eq.` (=), `.ne.` (≠), `.gt.` (>), `.ge.` (⩾), `.le.` (<), `.lt.` (⩽), `.and.`, `.or.`, `.not.`

```
program division
real*4 a,b,c
write(*,*) 'entrer la valeur de a et b'
real(*,*) a,b
if (b .eq. 0) then
    write(*,*) 'Division impossible !'
else
    c=a/b
    write(*,*) 'Le resultat est :', c
endif
end
```

6.2 Compteur boucle DO

```
do i=début,fin,pas
    instructions
enddo
```

N.B.: Par défaut pas=1. Dans tous les ca, il doit être un entier, positif ou négatif.

```
program factoriel
integer n,i
integer*4 fact
write(*,*) 'Entrer la valeur de n'
read(*,*) n
fact=1
do i=2,n
   fact=fact*i
enddo
write(*,*) 'Factoriel de n vaut :',fact
end
```

6.3 Boucle DO WHILE

```
do while (condition)
   instructions
enddo
```

7 Fichiers

Ouverture : `open`(`unit`=num, `file`='nom', `status`='statut')
avec num : numéro donné au fichier, 'nom' : nom du fichier, statut : 'old' ou 'new'
N.B.: num=5 : clavier ; num=6 : écran

Lecture : `read`(num,*)

Écriture : `write`(num,*)

Fermeture : `close`(num)

```
program tableau
integer i,j
real T(3,2)
open(unit=20, file='Tab.dat', status='new')
do i=1,3
   do j=1,2
      T(i,j)=i+j
      write(20,*) T(i,j)
   enddo
enddo
write(20,*) T( :, :)      ! chaque ' :' indique toutes les lignes (ou colonnes)
close(20)
end
```

Le fichier Tab.dat contient T(i,j) en colonne (car écriture dans la boucle) puis en lignes.

8 Formats de fichiers

Pour spécifier le format d'écriture ou de lecture : `write`(20,*) ou `read`(20,*), la deuxième étoile correspond au format.

- rFn.m : réel (float) de n chiffres dont m décimales, r est le facteur de répétition.
- rIn : entier (integer) de n chiffres significatifs
- An : chaine de n caractères
- rX : r blancs
- / : saut de ligne

```
      program vecteur
      integer i
      real*8 T(9)
      open(unit=20, file='Vect.dat', status='new')
      do i=1,9
         T(i)=float(i)/5
      write(20,100) T(i)      ! ou write(20,'(1X,F10.5)') T(i)
      enddo
100   format(1X,F10.5)
      close(20)
      end
```

Le ficher Vect.dat contient une colonne de valeurs de 10 chiffres significatifs dont 5 décimales.

```
      program tableau2
      integer i,j
      real T(3,2)
      open(unit=20, file='Tab2.dat', status='new')
      do i=1,3
         do j=1,2
            T(i,j)=i+j
         enddo
      enddo
      write(20,'(3F8.2)') T( :, :)
      close(20)
      end
```

Le ficher Tab2.dat contient une matrice de valeurs de 8 chiffres significatifs dont 2 décimales.

9 Fonctions et subroutines

Pour une meilleure lisiblité, le programme peut être séparé en plusieurs sous-parties. Celles-ci peuvent être écrites dans le même fichier du programme principale ou dans d'autres fichiers. Dans ce cas, ces fichiers devront être mentionnés lors de la compilation.

9.1 Fonctions

La première solution est representée par les fonctions. Elles sont déclarées par :

`<type sortie> function` nom(arg1,arg2,arg3,..)

avec arg1, arg2,... les variables d'entrée. Le type du résultat de la fonction doit être déclaré (p.ex. `integer`, `real`, etc.). Pour que le résultat soit retourné au programme principal, la fonction doit se terminer par `return`, puis `end`.
Pour appeler la fonction dans le programme principal, il ne faut pas oublier de déclarer la fonction comme une variable traditionnelle.

N.B.: Attention, toutes les variables doivent être réinitialisée dans les subroutines. Une seule variable en sortie est possible.

```
program factoriel3
integer n,m,i
integer fact     ! La fonction fact figure dans la liste des variables
integer factn,factm
write(*,*) 'entrer la valeur de n et m'
read(*,*) n,m
factn=fact(n)
factm=fact(m)
write(*,*) 'Factoriel de ', n, ' vaut :', factn
write(*,*) 'Factoriel de ', m, ' vaut :', factm
end

integer function fact(n)
integer n,i
fact=1
do i=2,n
   fact=fact*i
enddo
return
endfunction
```

9.2 Fonctions incorporées

`abs`, `mod`, `min`, `max`, `sqrt`, `exp`, `log`, `log10`, `cos`, `sin`, `tan`, `acos`, `asin`, `atan`, ...

9.3 Sous-programmes

Contrairement aux fonctions, les sous-programmes n'ont pas de limitations sur le nombre d'entrées/sorties. Ils sont déclarés par :

`subroutine` nom(arg1,arg2,arg3,...)

avec arg1, arg2,... les variables d'entrée et/ou de sortie. Ils se terminent par `return` - pour revenir au programme principale - et `end`.
Dans le programme principal, ils sont appelées par :

`call` nom(arg1,arg2,arg3,...)

N.B.: Comme pour les fonctions, toutes les variables doivent être réinitialisée dans les subroutines.

```
program factoriel2
integer n,i
integer*4 fact
call INIT(n)
call CALCUL(n,fact)      ! n en entree et fact en sortie
call PRINT(n,fact)       ! n et fact en entree
end

subroutine INIT(n)
integer n
write(*,*) 'Entrer la valeur de n'
read(*,*) n
return
end

subroutine CALCUL(n,fact)
integer n,i
integer*4 fact
fact=1
do i=2,n
   fact=fact*i
enddo
return
end

subroutine PRINT(n,fact)
integer n,i
integer*4 fact
write(*,*) 'Factoriel de',n,' vaut :',fact
return
end
```

9.3.1 Librairies disponibles

Outre les fonctions pré-existantes en Fortran, une panoplie de librairies contenant des fonctions plus compliquées est disponible. Certaines libraires sont payantes (NAG, etc.), d'autres sont ouvertes et gratuites (LAPACK, StatLib, etc.). Elles permettent d'élargir énormément les potentialités du Fortran. Pour pouvoir les utiliser une fois la ou les librairies téléchargées, il suffit de les inclure correctement lors de la compilation (d'abord il faut aussi les compiler, dans bien des cas). Voir `http://www.fortran.com/tools.html` pour une liste de librairies.

10 Compilation / exécution

Compiler un programme signifie créer, à partir de celui-ci, un fichier exécutable qui peut être fait tourner sur la machine. Lorsqu'on compile, les opération suivantes sont effectuées par le compilateur :

pre-processing : nettoyage et mise en forme des fichiers (p.ex., les commentaires sont éliminés).

optimization : le code peut être optimisé pour tourner plus rapidement, avec une occupation de mémoire moindre, etc.

compilation : les fichiers sont traduit en language machine. Des fichiers binaires dont l'extension est .o sont ainsi crées.

linking : les instructions venant des librairies (standards - c'est-à-dire celles propres au Fortran - et non - celle installées par l'utilisateurs) sont ajoutées. Enfin, le fichier exécutable est créé.

Plusieurs compilateur Fortran existent, avec des capacité différentes d'optimisation, par exemple, ou pouvant tourner sur différentes machines, etc. Nous on travaille avec gfrotran, un compilateur gratuit compatible avec tous les standards de Fortran jusqu'à Fortran95 et avec prise en charge partielle des Fortran2003 et Fortran2008. Pour compiler avec gfortran, dans un terminal taper

`gfortran <nom du programme>.f90 -o <nom de l'exécutable>.exe`

L'option `-o` suivi du nom du fichier executable impose un nom à l'exécutable. Autrement, par défaut il s'appelle `a.out`. Quant au compilateur, il en existe un certain nombre. Ici, nous utiliserons `gfortran`.

Pour exécuter le programme, taper

`./<nom de l'exécutable>.exe`

N.B.: Il faut compiler un programme à chaque fois qu'une modification est apportée au code !