Annexe A

Notes sur le langage Fortran 77

Contenu

A.1 Avant-propos

Ces notes sont un rappel très succinct sur le langage Fortran type 77 (avec quelques extensions reconnues par presque tous les compilateurs). Ceci représente un minimum pour commencer à programmer, mais le passage au Fortran 90 est fortement recommandé. Comme la compatibilité ascendante est quasi totale, ces notes ne sont utiles que pour rappeler des concepts de base et, comme nous le verrons dans le prochain chapitre, les extensions apportées par le Fortran 90.

A.2 La Structure de Base

Le découpage d'un programme en sous-programmes et fonctions représente le premier travail à réaliser avant de se lancer dans l'écriture du programme proprement dit.

Chaque partie du programme clairement définie est assignée à une tâche spécifique et doit être écrite selon la structure suivante : les données du sous-programme doivent être clairement séparées : données entrantes, données sortantes, données locales, fonctions de bibliothèque (locale ou non) utilisées. L'assemblage des différents éléments du programme est d'autant plus facile que les parties sont préalablement clairement écrites.

Le programme principal doit alors comporter un nombre limité d'instructions ce qui facilite à la fois la lecture du programme (utile pour le développeur et ceux qui consulteront ce programme) et surtout la mise au point, comportant généralement une phase non négligeable d'élimination d'erreurs (déboggage).

A.3 Règles minimales de programmation

Le langage Fortran est la contraction des mots anglais Formula Translation. Ce langage a été introduit dans les années cinquante et a subi plusieurs évolutions. Le Fortran 77, très ancien (le nombre fait référence à l'année du siècle dernier), présente une syntaxe qui présente des dangers pour avoir un contrôle sur la qualité du code écrit. Cela date de la création du langage et provient des contraintes initiales des ordinateurs qui ne disposaient alors que de peu de mémoire.

Le Fortran 90 représente une révision majeure du langage en introduisant des fonctionnalités manquantes au Fortran de la génération précédente. Le choix de garder une compatibilité quasi-totale avec le Fortran 77 présente des avantages à savoir que les programmes écrits depuis longtemps peuvent être compilés à nouveau par un compilateur de Fortran 90, mais aussi des inconvénients à savoir que l'écriture dangereuse des Fortran de vieille génération peut continuer à se pratiquer, même si les versions nouvelles permettent une meilleure écriture.

Le Fortran 95 apporte quelques fonctionnalités nouvelles, mais souligne aussi l'obsolescence de certaines instructions du langage Fortran des vieilles versions.

Le conservatisme des scientifiques a fait que le Fortran 77 est encore présent dans le domaine des sciences. Ce vieux Fortran possède tout de même quelques règles facultatives qui assurent une partie du contrôle de qualité du code. L'objectif de ces quelques notes est de montrer que, dans ce cadre, il est possible de produire un code qui présente un minimum de sécurité. Il est évident que, pour pouvoir disposer des fonctionnalités modernes (allocation dynamique de mémoire , structures, ...), il est nécessaire d'abandonner cette version du Fortran. Pour ceux qui connaissent le Fortran 90 et au-delà, la lecture de ce premier chapitre peut-être écourtée, mais elle permet de comprendre les différentes syntaxes rencontrées dans les programmes Fortran. Pour ceux qui ne connaissent que ce Fortran, ce chapitre doit permettre de pouvoir passer rapidement au Fortran 90.

Le langage Fortran 77, qui garde la compatibilité du Fortran 66, lequel date de l'époque de l'écriture de programmes sur cartes perforées a une contrainte de format pour l'écriture du code qui est la suivante

- Une instruction commence à partir de la colonne 7 et se finit au maximum à la colonne 72
- Une seule instruction par ligne (le retour à la ligne est a priori le séparateur d'instructions)
- Les commentaires du code sont écrits sur une ligne dont le caractère c a été placé à la première colonne
- Pour continuer une instruction qui dépasse la colonne 72, on place un caractère en colonne 6 sur la ligne suivante et on continue à écrire l'instruction à partir de la colonne jusqu'à la colonne 72. On peut continuer cette règle sur plusieurs lignes si nécessaire.

Les règles minimales que l'on adopte sont les suivantes :

1. Placer la directive **implicit none** dans tous les sous-programmes, fonctions, et programme principal. Cela oblige à déclarer toutes les variables en précisant de plus leur type. Cette règle qui peut paraître contraignante est indispensable pour assurer un minimum de sécurité dans l'écriture d'un programme[1].
2. Toutes les variables utilisées doivent déclarées.
3. Toutes les fonctions utilisées doivent être déclarées.
4. Les étiquettes sont interdites hormis celles pour le format. (On peut toujours s'en passer, voir ci-dessous)
5. Malgré la "limitation" de la colonne 72, indenter les programmes, cela accroît considérablement la lisibilité de ceux-ci.

A.4 Les déclarations de variables

Le langage Fortran est un langage fortement typé, car il possède un grand nombre de types prédéfinis.

A.4.1 Types prédéfinis

Type	Taille	borne supérieure	borne inférieure
integer	entier 4 octets	-2147483648	2147483647
real	réel de 4 octets	-3.4×10^{38}	3.4×10^{38}
real *8	réel de 8 octets	-1.7×10^{308}	1.7×10^{308}
character	caractère de 1 octet		
boolean	1 octet		
complex	2 réels de 4 octets		
complex *8	2 réels de 8 octets		

[1] Il est clair que pour des programmes de moins de 100 lignes, on pourrait se dispenser de cette contrainte car on peut assez contrôler à la main le code produit. Pour des programmes de simulation numérique, le nombre de lignes de code varie généralement entre plusieurs centaines et plusieurs dizaines de milliers actuellement, ce qui conduit à privilégier une écriture propre avec un langage qui le permet.

Une variable logique peut-être uniquement vraie ou fausse. La syntaxe est

```
a=.true.
a=.false.
```

L'assignation d'une valeur pour une variable complexe est de la forme

```
b=cmplex(2.0,1.0)
```

où plus simplement

```
b=(2.0,1.0)
```

L'initialisation des variables peut se faire avec l'instruction data

```
integer *4 i
real *4 x
data i/7/,x/4.5/
```

A.4.2 Les tableaux

En Fortran les tableaux commencent par défaut avec un indice égal à 1. La déclaration d'un tableau de 100 éléments de réels à double précision est

```
real *8 a(100)
```

Le premier élément est a(1) et le dernier a(100). Noter qu'en Fortran, les parenthèses servent à la fois pour les tableaux et les arguments de fonctions, ce qui ne facilite pas la lecture d'un programme et nécessite d'autant plus d'être rigoureux dans l'écriture.

Il est possible de faire démarrer un tableau avec un indice 0 avec la déclaration suivante

```
real *8 a(0:100)
```

le tableau contient alors 101 éléments et a(0) est le premier élément. On peut utiliser un entier négatif pour le premier élément. Une telle fonctionnalité peut s'avérer pratique.

Les tableaux à plusieurs dimensions sont définis en Fortran de la manière suivante

```
real *8 a(10,20)
```

A noter que contrairement au langage C qui stockent les données du tableau par ligne comme nous verrons dans le chapitre sur le langage C, les tableaux sont stockés en mémoire en colonnes.

L'initialisation des tableaux peut se faire aussi avec l'instruction data

```
real *8 a(10,20)
real *8 b(2,2)
data a/ 200 * 0.0/
data b/3.0,0.7,-4.1,1.0/
```

Rappelons que, pour la matrice b, l'affectation des éléments se fait selon l'ordre des colonnes.

A.5 Fonctions et sous-programmes

A.5.1 Définitions

Le Fortran fait une distinction (contrairement aux langage C et C++) entre les fonctions qui ont un type (en l'occurrence pour le fortran 77, ce type est un type prédéfini) et les sous-programmes, qui n'ont pas de type. Les fonctions s'appellent *function* et les sous-programmes *subroutine*. Nous donnons deux exemples très simples pour illustrer les règles d'utilisation de ces notions. Soit une fonction réalisant le carré d'un nombre réel

```
program toto
implicit none
real a,b
real sqr
external sqr
a=2.0
b=sqr(a)
write(*,*)a,b
end

function sqr(a)
implicit none
real sqr
real a
sqr=a**2
return
end
```

Le même programme avec cette fois-ci l'élévation au carré défini à partir d'un sous-programme. Le paramètre du sous-programme joue le rôle à la fois de variable d'entrée et de variable de sortie.

```
program toto
implicit none
real a
a=2.0
write(*,*)a
call sqr_2(a)
write(*,*)a
end

subroutine sqr_2(a)
implicit none
real a
a=a**2
return
end
```

A.5.2 Instructions : common et parameter

Un bloc **common** est une instruction qui permet de stocker des données préalablement définies dans une liste : celle-ci peut être utilisée dans différents sous-programmes. La syntaxe est la suivante

```
real *8 b,c
integer n,j
real *8 a(2,3)
common/poubelle/a,b,c,n
```

Le nom du common est ici poubelle, et en rappelant la dernière instruction dans un sous-programme ou une fonction, on peut utiliser toutes les variables définies dans le bloc **common**.

L'utilisation des blocs **common** est à proscrire. Une partie des blocs common peut être supprimée en donnant des arguments au sous-programmes et fonctions du programme.

Un des risques d'erreurs en Fortran (non signalé par le compilateur) est de ne pas recopier à l'identique la déclaration du bloc common dans le sous-programme. Par exemple

```
program toto
implicit none
integer a(20)
integer ini,n
common/initia/ini,n,a
integer i
n=20
ini=10
call init
do i=1,n
   write(*,*)' ',i,a(i)
enddo
end
subroutine init
implicit none
integer a(20)
common/initia/n,ini,a
do i=1,n
   a(i)=ini
enddo
return
end
```

Ce programme initialise à 20 les dix premiers éléments du tableau a, alors que l'on souhaite initialiser à 10 les 20 éléments du tableau. L'échange de l'ordre des variables *ini* et n dans la déclaration du bloc common est à l'origine du résultat

non désiré. Cette erreur classique est parfois très difficile à détecter dans un programme comportant un grand nombre de lignes et de sous-programmes.

On peut réécrire ce programme de manière juste et claire en se passant du bloc common.

```
program toto
implicit none
integer a(20)
integer ini,n
integer i
n=20
ini=10
call init(a,ini,n)
do i=1,n
   write(*,*)' ',i,a(i)
enddo
end
subroutine init(a,ini,n)
implicit none
integer a(n)
do i=1,n
   a(i)=ini
enddo
return
end
```

L'instruction **parameter** permet de déclarer des constantes dans un sous-programme, fonction ou programme principal où celles-ci ont été préalablement définiees. On peut définir des constantes à partir de constantes déjà définies. Par exemple :

```
integer n,m
parameter (n=20,m=4*n)
```

A.6 Les instructions itératives

Comme il arrive souvent d'avoir à faire des opérations répétitives, on dispose en programmation d'instructions qui vont entourer un bloc d'instructions pour les faire exécuter un grand nombre de fois. Si le nombre est connu à l'avance, on utilise l'instruction **do enddo** qui contient un indice de boucle (variable entière).

```
integer m,n,step
do i=m,n
...
enddo
```

où step correspond au pas d'incrémentation. Si n est strictement inférieur à m, aucune instruction de la boucle ne sera exécutée. Si le nombre d'itérations dépend d'une condition, la syntaxe est

```
do while(condition)
...
enddo
```

Attention : Une boucle while n'est interrompue que si la condition devient fausse en un nombre d'itérations fini. Parmi les erreurs classiques de programmation, il arrive qu'un programme n'arrive pas à se finir car la condition reste indéfiniment vraie

A.7 Les instructions conditionnelles

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

Cette instruction signifie que si la condition est vraie les instructions du groupe 1 sont exécutées, tandis que si la condition est fausse les instructions du groupe 2 sont exécutées.

On peut imbriquer des instructions if, par exemple,

```
if (condition1) then
      instructions1
   elseif (condition2) then
      instructions2
   else
      instructions3
   endif
```

Ce groupe signifie que si les deux conditions sont vraies, les instructions 2 sont exécutées ; si la condition 1 est vraie (indépendamment du statut de la condition 2), les instructions 1 sont exécutées ; si la condition 1 est fausse et la condition 2 vraie, les instructions 2 sont exécutées ; si les deux conditions sont fausses, rien n'est exécuté..

A.8 Opérateur de comparaison, opérateurs d'assignation

Pour formuler une expression logique, il est souvent nécessaire d'avoir un opérateur de comparaison, le langage Fortran 77 fournit plusieurs opérateurs que nous rassemblons ci-dessous

.*LT*. inférieur(strictement) à
.*GT*. supérieur (strictement) à
.*LE*. inférieur ou égal à
.*GE*. supérieur ou égal à
.*NE*. différent de
.*EQ*. égal à

Il existe aussi des opérateurs de comparaison logiques

.*AND*. et logique
.*OR*. ou logique
.*XOR*. ou exclusif

Par exemple, soit deux variables booléennes et une variable entière, on peut placer l'instruction logique dans un **if** où un **while**.

```
integer i
boolean a,b
...
if(((a.OR.b)).AND.(if<5)) then
     ...
    endif
```

A.9 Les entrées, les sorties, les formats

Pour lire des données au clavier, on peut utiliser le format par défaut suivant

```
read (*,*) variable
```

Pour écrire à l'écran, on utilise

```
write(*,*) variable
```

Il exite aussi l'instruction **print** dans le style obsolète !

On peut aussi écrire en alternant les chaines de caractères et variables

```
write (*,*) ' var1= 'variable1,'var2 =',variable2
```

Si l'on souhaire écrire avec un format bien spécifique on peut écrire l'instruction suivante pour deux variables une entière et une réelle.

```
integer i
real a
 write(*,'(I5 E15.5)')i,a
```

Les instructions pour le format sont les suivantes

A chaîne de caractères
D nombre en double precision et notation exponentielle
E nombre réel et notation exponentielle
F nombre réel et notation flottante
I entier
X espace horizontal
/ retour à la ligne

Le format 2I5 est pour deux entiers écrits avec 5 caractères. Le format E15.5 est pour un nombre réel en notation exponentielle écrit sur quinze caractères dont 5 après la virgule.

Pour lire sur un fichier, il faut préalablement ouvrir ce fichier et donc définir un descripteur de fichier qui, en Fortran, est un numéro.

Exemple :

```
open(60,file='gr.res')
```

signifie que l'on ouvre le fichier gr.res qui se trouve dans le répertoire où le programme est exécuté et que ce fichier est un fichier texte (c'est à dire formaté).

Pour lire les données, on utilise un ordre

```
real *8 variable2,variable3
read(60,'(2E15.5))') variable2, variable3
```

signifie que l'on deux réels à lire avec un format de 15 caractères à 5 décimales.

Une fois réalisé l'ensemble des opérations d'écriture, on ferme un fichier en indiquant son numéro de descripteur

```
close(60)
```

A.10 Les fonctions intrinsèques

Le langage Fortran possède un nombre significatif de fonctions intrinsèques dans la bibliothèque standard. Il en existe principalement de deux types : les fonctions qui permettent de convertir les données d'un type en autre type (par exemple entier en flottant) et les fonctions mathématiques, à la fois les fonctions transcendantes usuelles et une partie des fonctions spéciales rencontrées assez fréquemment.

A.10.1 Les conversions de type

La deuxième colonne indique le type de la variable attendue comme argument de la fonction et la troisième colonne le type de la fonction.

Pour la deuxième et troisième colonne des tableaux ci-dessous, le I signifie entier, le R signifie réel simple précision, le D réel double précision et X le complexe

AIMAG	X	R	partie imaginaire d'un nombre complexe
AINT	DR	DR	tronque la partie fractionnaire en préservant le type
ANINT	DR	DR	arrondit au nombre entier le plus proche en préservant le type
CMPLX	DIRX	X	convertit un nombre en complexe
DBLE	DIRX	D	convertit en double précision
INT	DIRX	I	convertit en entier par troncation
NINT	DR	I	convertit en entier par arrondi
REAL	X	R	partie réelle d'un nombre complexe
REAL	DIR	R	convertit en réel

A.10.2 Fonctions mathématiques

Les fonction mathématiques les plus standards sont accessibles par défaut dans la bibliothèque Fortran.

ABS	X	R	module d'un nombre complexe
ABS	DIR	DIR	valeur absolue d'un nombre
ACOS	DR	DR	arccos d'un nombre
ASIN	DR	DR	arcsin d'un nombre
ATAN	DR	DR	arc-tangente d'un nombre
CONJG	X	X	complexe conjugué d'un nombre complexe
COS	DRX	DRX	cosinus d'un nombre
EXP	DRX	DRX	exponentielle
LOG	DRX	DRX	logarithme néperien
LOG10	DRX	DRX	logarithme décimal
MAX	DIR,DIR	,...	DIR valeur maximale
MIN	DIR,DIR	,...	DIR valeur minimale
MOD	DIR	DIR	arg1 modulo arg2
SIN	DRX	DRX	sinus d'un nombre
COSH	DR	DR	cosinus hyperbolique d'un nombre
SINH	DR	DR	sinus hyperbolique
SQRT	DRX	DRX	racine carré
TAN	DR	DR	tangente d'un nombre
TANH	DR	DR	tangente hyperbolique

A.11 Passage des variables, tableaux, et fonctions

Le passage des variables du Fortran 77 est simple, mais aussi dangereux, puisqu'il s'agit d'un passage par référence. (C'est évidemment une amélioration considérable par rapport au common.)

Ainsi seuls les variables locales du sous-programme ou de la fonction ne sont pas connues du programme principal. Pour le passage des tableaux, il suffit de transmettre le nom du tableau pour avoir accès à ce tableau dans le sous-programme et de déclarer le tableau dans le sous-programme avec une dimension variable.

Pour les fonctions, il est nécessaire de déclarer la fonction appelée avec l'ordre external et de préciser son type. Le nombre de variables de la fonction appelée n'a pas besoin d'être spécifié. Cela peut être bien entendu un peu dangereux, puisque aucune vérification n'est faite à la compilation. Le programme suivant calcule deux integrales par la méthode des trapèzes ; une version dans chaque langage est donnée dans les autres chapitres. précédent.

```
program integrale
implicit none
integer  n_point
real *8 a_inf,b_sup
real *8 rectangle,trapeze,simpson
real *8 func_1,func_2
real *8 res,res_ex
```

```
      integer nf
      external rectangle,trapeze,simpson
      external func_1,func_2,func_3
c     variables locales
      integer k
c*****************************************************************
c      premiere fonction
c*****************************************************************
      a_inf=0.0
      b_sup=0.5
      n_point=10
      res_ex=1.0d0/3.0d0+log(3.0d0)/4.0d0
      write(*,*)' Iex=',res_ex
c*****************************************************************
         res=trapeze(a_inf,b_sup,n_point,func_1)
c*****************************************************************
c      deuxieme fonction
c*****************************************************************
      b_sup=1.0
      n_point=10
      res_ex=acos(-1.0d0)/4.0d0
      write(*,*)' Iex=',res_ex

      res=trapeze(a_inf,b_sup,n_point,func_2)
c*****************************************************************
      end
c fin du programme principal
c*****************************************************************

c*****************************************************************
      function func_1(x)
      implicit none
      real *8 func_1,x
      func_1=1.0/((1.0-x*x)*(1.0-x*x))
      return
      end
c*****************************************************************
      function func_2(x)
      implicit none
      real *8 func_2,x
      func_2=1.0/(1.0+x*x)
      return
      end
c*****************************************************************
      function trapeze(a,b,n,f)
      implicit none
      real *8 trapeze,a,b,f
```

```
      integer n
c     variables locales
      real *8 sum,delta
      integer i
      external f
c     calcul par les trapezes
      sum=0.0
      delta=(b-a)/real(n)
      do i=1,n-1
         sum=sum+f(a+delta*i)
      enddo
      trapeze=delta*(sum+0.5*(f(a)+f(b)))
      return
      end
c*****************************************************************
```

A.12 Conclusion et références

Ces quelques notes ont pour but de pouvoir de donner quelques règles pour l'écriture des programmes et de renvoyer à des ouvrages plus spécialisés. Le Fortran 77 souffre d'un nombre de limitations, en sus des défauts précédemment énoncés sur lesquels nous ne revenons pas. Les règles énoncées ci-dessous permettent de commencer à envisager une programmation plus rigoureuse qui ne peut être faite qu'avec un langage plus adapté, ce que nous verrons dans le chapître suivant.

Pour avoir une documentation en ligne sur le Fortran 77, on peut consulter le site http ://www.enstimac.fr/ gaborit/lang/CoursDeFortran/

Annexe B

Interfacer du Fortran et C

B.1 Introduction

Il arrive parfois d'avoir besoin de combiner des morceaux de code écrits dans des langages différents ; Il est possible en effet de mélanger à condition de disposer d'un compilateur dans les différents langages. Comme nous allons le voir ci-dessous cette possibilité nécessite quelques précautions. Nous allons illustrer ces possiblités sur l'appel de fonctions

B.2 Le langage Fortran 77 fait appel au langage C

Considérons tout d'abord le langage Fortran 77 qui va inclure du langage C. Il faut donc compiler le fichier du langage C sans faire l'édition de liens. Cela revient à créer un fichier objet. Sur la plupart des compilateurs, cela revient à mettre l'option -c.

```
float  sqr_(float *a)
{
  return((*a)*(*a));}
```

Le reste du programme en Fortran est donné par le programme

```
      program toto
       real a,b
       real sqr
       external sqr
       a=4.0
       b=sqr(a)
       write(*,*)a,b
       end
```

Il faut compiler le programme en utilisant le fichier objet au moment de l'édition de liens.

On note deux points importants : le seul passage des variables en Fortran 77 est le passage par référence, d'où le fait que l'on a un pointeur dans la fonction

écrite en C. Pour la correspondance des noms entre la fonction en Fortran et celle en C, on note l'ajout du caractère souligné dans la déclaration de la fonction en C. Ce changement, bien que classique, dépend du compilateur utilisé. Cela fonctionne avec le compilateur GNU et le compilateur Intel.

Dans le cas d'un sous-programme en Fortran, on doit avoir une fonction de type void en C.

B.3 Le langage Fortran 90 fait appel au langage C

Dans cet exemple, nous faisons le choix de créer une fonction dont la variable donnée en argument qui est une variable d'entrée. On définit aussi le prototype de la fonction en Fortran 90.

```
program toto
  real :: a,b
  interface
     real function  sqr ( a)
       real, intent(in)::  a
     end function sqr
  end interface
  a=4.0
  b=sqr(a)
  write(*,*)a,b
end program toto
```

Contrairement à ce que l'intention pourrait laisser croire le fortran 90 continue de passer les variables par adresse et non pas par valeur même lorsqu'il s'agit de variables simples. En compilant avec le morceau de code en C, on obtient le résultat correct.

Il faut ajouter que le mélange de code peut être une source d'erreur ; en effet, dans le langage Fortran en créant le prototype de la fonction ; on garantit le code a priori. En fait le prototypage ne se transporte pas au changement de langage, car la vérification se fait par compilateur. Aucune vérification se fera sur le compilateur C, a moins de faire un prototype similaire. Pour vérfier ce propos, on peut tester en compilant la fonction C modifiée suivante

```
float  sqr_(float *a)
{ *a=10;
  return((*a)*(*a));}
```

On a alors un effet de bord désastreux.

On voit donc que les mélanges de langage sont techniquement possibles, mais leur utilisation reste un peu délicate.

Table des matières