Eléments de Fortran 90

Guy Munhoven

September 18, 2011

Introduction

Il existe un grand nombre de cours d'introduction à Fortran 90 sur le WWW, dont certains sont excellents. Je n'en citerai que les trois suivants:

- `http://perso.fundp.ac.be/~amayer/cours/Fortran90/index.html`
- `http://www.obs.u-bordeaux1.fr/radio/JMHure/CoursF95CMichaut.pdf`
- `http://www.idris.fr/data/cours/lang/fortran/choix_doc.html`

Cette brève introduction est basée pour l'essentiel sur dernier de la liste ci-dessus, développé à l'IDRIS (Institut du Développement et des Ressources en Informatique Scientifique), un institut du CNRS localisé à Orsay. Le lien ci-dessus offre ce cours en plusieurs déclinaisons et formats de fichier.

De plus, les trois ouvrages suivants m'ont été particulièrement utiles :

- Gehrke W. (1995) *Fortran 90 Language Guide*. Springer-Verlag, Berlin, 385 pp.
- Redwine C. (1995) *Upgrading to Fortran 90*. Springer-Verlag, New York, 416 pp.
- Chivers I. D. and J. Sleightholme (2006) *Introduction to Programming with Fortran. With Coverage of Fortran 90, 95, 2003, and 77*. Springer-Verlag, London, 592 pp. `http://dx.doi.org/10.1007/b137984`

Petit historique

1954-57 Développement d'un langage symbolique (différent de code machine et de l'assembleur) appelé FORTRAN (pour *FORmula TRANslation*) par une équipe de chez IBM dirigée par John Backus

1957 Premier compilateur disponible

1958 FORTRAN II: sous-programmes compilables indépendamment

1967 FORTRAN IV (FORTRAN 66)

1978 FORTRAN V (FORTRAN 77): introduction du type `CHARACTER`, de la structure `IF-THEN-ELSE`, traitement

1991 Fortran 90: modernisation de FORTRAN 77, standardisation d'un bon nombre d'extensions propres aux constructeurs de compilateurs, introduction de concepts de la programmation orientée objet, et, nouveau format de code source

1996 Fortran 95 (peu de différences avec Fortran 90)

2004 Fortran 2003

Chapter 1

Programme Fortran 90: structure générale et syntaxe

1.1 Jeu de caractères admis en Fortran 90

Les caractères suivants peuvent être utilisés dans un code source Fortran 90 :

- lettres `A`–`Z`, `a`–`z`, chiffres `0`–`9`
- caractères spéciaux: `= + - * / ! ? : . ; , " ' ( < > ) % & $`
- le caractère espace (blanc)
- le caractère _ (*underscore*)

Le contenu de chaînes de caractères peut contenir d'autres caractères encore. A noter que le Fortran 90 standard ne distingue pas entre minuscules des majuscules, hormis pour le contenu de chaînes de caractères.

1.2 Format

Il existe deux formats de code : le format *libre* et le format *fixe*.

Le format libre permet des lignes longues de 132 caractères. Des instructions plus longues peuvent être codées sur plusieurs lignes, en terminant chaque ligne incomplète par le caractère de continuation `&`. Lorsqu'une chaîne de caractères doit être coupée, le premier caractère significatif sur la ligne de continuation doit impérativement être précédé d'un `&` aussi. Il est permis d'avoir jusqu'à 39 lignes de continuation.

Le format fixe représente un héritage des standards FORTRAN antérieurs à Fortran 90, datant du temps des cartes perforées. Même s'il est toujours

compatible avec Fortran 90, il est recommandé de ne plus l'utiliser. Comme vous pourriez probablement encore être confronté à des codes sous ce format, en voici une courte description. Chaque ligne comporte 3 zones différentes:

1. colonnes 1–5: réservées pour contenir une étiquette (*label*), c'est-à-dire, une suite de 1 à 5 chiffres non tous nuls, servant au repérage et jalonnage du programme

2. colonne 6: un caractère non-blanc dans cette colonne indique que cette ligne est la continuation de la ligne précédente—dans ce cas, les colonnes 1 à 5 doivent rester vides

3. colonnes 7–72: instructions et mots-clé

En FORTRAN 77 standard, il peut y avoir jusqu'à 9 lignes de continuation consécutives. Sous Fortran 90 standard, en format fixe, il peut y avoir jusqu'à 19 lignes de continuation consécutives (contre 39 en format libre)!

En Fortran 90, plusieurs commandes peuvent être mises sur une ligne. Elles sont alors séparées les unes des autres par des `;`.

1.3 Commentaires

En format libre le caractère `!` indique que la suite de cette ligne est un commentaire et ne doit pas être traitée par le compilateur. Il est permis de n'avoir rien d'autre qu'un commentaire sur une ligne. Il est recommandé de faire amplement appel à cette facilité que représentent les commentaire, afin de bien documenter des déroulements de programmes.

En format fixe, les lignes de commentaires sont caractérisées par un `c`, `C`, une `*` en première colonne, ou un caractère `!` ailleurs qu'en sixième colonne. A cet endroit, un caractère `!` continue à signaler une ligne de continuation. A noter que le standard FORTRAN 77 ne permet pas d'utiliser le caractère `!` pour commencer un commentaire. Enfin, en format fixe, une ligne de commentaire ne peut pas être continuée par un caractère non-blanc en sixième colonne de la ligne suivante. Chaque continuation d'une ligne de commentaires doit être considérée comme ligne de commentaires propre. Il est clair que le caractère `!` perd sa signification de début de commentaire s'il fait partie d'une chaîne de caractères.

Le standard Fortran 90 prévoit une restriction importante quant au placement d'un commentaire dans un code source en format libre: un `!` ne peut pas suivre un `&` qui scinde une chaîne de caractères. Certains compilateurs (notamment le `pgf90` de Portland Group, utilisé, entre autres, sur le cluster de calcul intensif du SeGI de l'ULg) acceptent des commentaires à ces

endroits, ouvrant ainsi la porte à des codes non-standard, et non portables. D'autres compilateurs, comme p.ex., `ifort` de Intel et le `gnu gfortran` sont plus stricts à cet égard et ne compileront pas de tels coses.

En général, les fichiers source ayant une extension `.f`, `.F` ou `.for` sont supposés être en format fixe, les fichiers à extension `.F` devant en plus être pré-compilés. Les fichiers source ayant une extension `.f90` ou `.F90` sont supposés être en format libre, ceux en `.F90` doivent encore d'être précompilés (à l'aide du pré-compilateur C, ou d'un autre pré-compilateur comme `fpp`). Ceci permet d'inclure des directives de précompilation dans le code, qui seront évaluées par le précompilateur, afin de produire des versions particulières du code source (incluant, p.ex., des instructions supplémentaires, des versions spéciales, etc.). Généralement, le compilateur Fortran appelle automatiquement un pré-compilateur approprié en cas de besoin, et prend en charge la gestion des fichiers temporaires intermédiaires éventuels.

1.4 Structure générale d'un programme Fortran 90

Fortran 90 distingue les unités de programme (*scoping units*) suivantes :

- programme principal;
- sous-programmes: sous-routines et fonctions;
- modules;
- *block data*.

Dans un premier temps, nous ne nous occuperons pas des modules, ni des *block data*. Il faut cependant noter que les modules comptent parmi les innovations les plus utiles de Fortran 90.

1.4.1 Programme principal

Le programme principal a la structure suivante :

```
[PROGRAM [nom_du_programme]]
   instructions de spécification et de déclaration
   instructions exécutables
[CONTAINS
   procédures internes]
END [PROGRAM [nom_du_programme]]
```

Tout ce qui est mis entre crochets ([...]) est optionnel (les crochets eux-mêmes sont à omettre dans tous les cas). L'instruction `PROGRAM` n'est pas obligatoire, mais bien l'instruction `END`. Cette dernière peut être complétée par l'attribut `PROGRAM` qui doit alors être suivie du nom du programme *nom_du_programme*. Les différents types de contenu (instructions de spécification et de déclaration, instructions exécutables, etc.) sont décrits ci-dessous.

1.4.2 Sous-routines

Les sous-programmes de type sous-routine ont une structure semblable à celles du programme principal :

```
SUBROUTINE nom_de_la_sous-routine [(arguments)]
   instructions de spécification et de déclaration
   instructions exécutables
[CONTAINS
   procédures internes]
END [SUBROUTINE nom_de_la_sous-routine]
```

L'instruction `END` est obligatoire, mais non pas la spécification supplémentaire `SUBROUTINE`. Si elle est donnée, ce qui est recommandé, surtout au cas où plusieurs sous-programmes sont réunies dans un même fichier, elle doit être complétée par le nom du sous-programme, exactement comme spécifé à la ligne `SUBROUTINE`.

Les sous-routines sont exécutées à l'aide de la commande

```
CALL nom_de_la_sous_routine [(arguments)]
```

1.4.3 Fonctions

Les sous-programmes de type fonction retournent un résultat en les appelant par leur nom :

```
[type] FUNCTION nom_de_la_fonction ([liste_d'arguments])
   instructions de spécification et de déclaration
   instructions exécutables
[CONTAINS
   procédures internes]
END [FUNCTION nom_de_la_fonction]
```

Le type (voir ci-dessous) peut être spécifié (recommandé); sinon, il est déterminé par les règles de typage par défaut (voir ci-dessous).

1.4.4 Procédures internes

Les trois types de programmes et sous-programmes peuvent contenir l'instruction `CONTAINS`. Cette instruction sépare la dernière instruction exécutable du programme en question et marque le début d'une collection de sous-programmes (sous-routines ou fonctions), appelées procédures internes. Contrairement aux sous-programmes externes, c'est-à-dire qui ne sont pas localisés à l'intérieur d'un programme ou sous-programme, ces sous-programmes ne sont utilisables qu'à l'intérieur du programme qui les contient. De plus, ces procédures internes ont accès à toutes les variables du programme qui les contient (le programme-hôte), sauf si elles incluent des déclarations d'identificateurs qui portent des noms d'identificateurs utilisés dans le programme hôte.

Ces procédures internes permettent d'un côté une meilleure structuration des tâches à effectuer; d'un autre côté le partage des variables peut s'avérer dangereux, puisqu'il n'y as moyen d'empêcher les sous-routines internes de modifier le contenu de variables du programme principal.

Chapter 2

Déclarations

2.1 Identificateurs: noms possibles

Un identificateur permet de donner un nom à :

- une variable,
- une constante,
- un sous-programme (sous-routine ou fonction).

Un identificateur doit répondre aux spécifications suivantes :

- il est défini par une suite de caractères alphanumériques (lettres non accentuées, chiffres, *underscore*),
- son premier caractère doit être une lettre,
- sa longueur est limitée à 31 caractères,
- on ne distingue pas les lettres majuscules des minuscules.

En conséquence: ni blancs, ni caractères spéciaux dans les identificateurs.

2.2 Types de variables

Le standard Fortran 90 connaît cinq types de variables de base[1], ainsi que le mot-clé `TYPE` qui permet de définir des types dérivés des types de base (p.ex., des structures). Dans cette première introduction, nous nous limiterons aux

[1]Dans le standard Fortran 90, `DOUBLE PRECISION` est considéré comme un genre particulier de `REAL`.

types de base. Les caractéristiques (espace mémoire requis par unité, valeurs possibles) sont donnés au tableau 2.1. A noter que certaines caractéristiques (p.ex., si le type `INTEGER` par défaut correspond au type à 2 ou à 4 octets) dépendent du compilateur utilisé. Fortran 90 offre cependant des possiblilités pour permettre un contrôle précis du type désiré (via l'attribut `(KIND=`*nbr*`)`, et les fonctions intrinsèques

`SELECTED_INT_KIND(`*puissance_de_10_du_maximum*`)`

ou encore

`SELECTED_REAL_KIND(`*puissance_de_10_du_maximum*`)`.

Pour plus de détails concernant ces possibilités très puissantes de déclaration de variables, vous pourrez consulter l'un des cours précités. Implicitement, toutes les variables sont de type `REAL`, hormis celles dont les identificateurs commencent par `I`, `J`, `K`, `L`, `M`, ou `N`, qui sont de type `INTEGER`.

S'il est vrai que ce typage par défaut permet la facilité de ne pas devoir déclarer toutes les variables, il est néanmoins à l'origine de beaucoup d'erreurs de programmation. Il est fortement déconseillé d'y faire appel. Il vaut mieux le désactiver par l'instruction `IMPLICIT NONE` avant toute déclaration.

2.3 Déclarations : forme générale

Les déclarations se mettent sous la forme générale suivante :

type [, *attributs*] [`::`] identificateur [`=` *valeur_initiale*]

Dans cette déclaration, *type* est l'un des types du tableau 2.1 (ou alors un type dérivé). Les attributs permettent de conférer des propriétés particulières à des variables. Les attributs les plus courants sont les suivants :

- `PARAMETER` rend la variable non modifiable (constante) ;
- `DIMENSION(`*dimensions*`)` en fait un tableau avec des dimensions données[2]

[2]Il est important de ne pas confondre les *tableaux* (*arrays* en Anglais) et les *matrices* (*matrix*, pl. *matrices*, en Anglais). Si l'on désire interpréter un tableau comme matrice, il faudra explicitement le faire. Ainsi, p. ex., les opérations arithmétiques entre tableaux sont bien définis, mais opèrent composante par composante. Pour l'addition de deux tableaux de dimensions compatibles, le résultat correspond bien à l'addition des matrices correspondantes. Pour la multiplication, cependant cela n'est pas vrai, et il faut faire appel à des fonctions particulières pour réaliser les opérations du calcul matriciel. Sous MATLAB ou OCTAVE, par contre, les variables bi-dimensionnelles sont en premier lieu des matrices et l'aritmétique de tableau (c'est-à-dire, composante par composante) demande des opérateurs particuliers (comme, p. ex., `.*` pour la multiplication).

Table 2.1: Types de variables standard Fortran 90

Type	Octets/unité	Etendue	Précision
`LOGICAL`	1–4	`.TRUE.`, `.FALSE.`	—
`INTEGER`	2	$-32768 < i < 32767$	exacte
	4	$-2.147.483.648 < i < 2.147.483.647$	exacte
`REAL`	4	$1.2 * 10^{-38} < \|r\| < 3.4 * 10^{+38}$	6–7
`DOUBLE PRECISION`	8	$2.2 * 10^{-308} < \|r\| < 1.8 * 10^{+308}$	15–16
`COMPLEX`	2*4	voir `REAL`	
`CHARACTER`	1/caractère	—	—

- `SAVE` la rend statique (en Fortran 90 standard, toutes les variables d'un sous-programme deviennent indéfinies à la sortie d'un sous-programme—avec `SAVE`, elles préserveront leurs valeurs actuelles qui pourront être utilisées au prochain appel) ;
- `INTENT(IN)`, `INTENT(OUT)` et `INTENT(INOUT)` ne peuvent être appliqués que dans des sous-programmes, et là, uniquement aux identificateurs de variables faisant partie de la liste d'arguments. Ils permettent de limiter l'utilisation des variables concernées de la manière suivante :
  - `INTENT(IN)` : le contenu de cette variable est destiné uniquement à être lu et ne peut pas être modifié (toute tentative de modification sera sanctionnée par une erreur à la compilation),
  - `INTENT(OUT)` : cette variable est destinée à recevoir un contenu à l'intérieur du sous-programme et n'est pas encore initialisé (toute lecture de contenu d'une telle variable avant initialisation dans l'unité de programme en question donnera lieu à une erreur de compilation)
  - `INTENT(INOUT)` : cette variable peut être considérée comme initialisée et son contenu comme librement modifiable.
- `ALLOCATABLE` qui définit la variable comme alouable de manière dynamique

Une variable de type `CHARACTER` demande en outre que l'on spécifie sa longueur maximale, de préférence sous la forme

`CHARACTER(LEN=`*longueur_max*`) ::` *charvar_name*

Afin d'éviter des surpises, il est hautement recommandé de faire appel à la possibilité d'initialiser les variables à la déclaration, soit à des valeurs par défaut, soit à des valeurs considérées comme illégales dans le programme (p.ex., une concentration négative, un nombre d'objets négatif, ...) et de faire en sorte que le programme va générer des erreurs lorsque ces valeurs sont utilisées. A noter que l'initialisation d'une variable lors de la déclaration

- lui confère automatiquement l'attribut `SAVE` ;
- n'est prise en compte que jusqu'à la première modification de son contenu; dans le cas d'une variable dans une sous-routine, lors d'un deuxième appel à la sous-routine, les variables en question vont avoir le même contenu qu'à la fin de l'appel précédent.

Pour une liste détaillée de tous les attributs possibles, veuillez consulter les cours indiqués ci-dessus.

Chapter 3

Opérateurs

Les opérateurs pour variables numériques offerts par Fortran 90 sont repris au tableau 3.1. S'y ajoute encore l'opérateur de concatenation (`//`) pour les chaînes de caractères. Les expressions complexes formées à l'aide de ces opérateurs se notent sous forme algébrique. Il y a des règles de précédence des opérateurs les uns sur les autres qui sont proche de ce qui est utilisé habituellement en mathématiques. Ainsi, l'exponentiation a priorité sur la multiplication qui a priorité sur l'addition. En cas de doute, il est recommandé d'utiliser des parenthèses pour structurer les expressions et lever toute ambigüité.

Les opérateurs relationnels servent à comparer des variables numériques, et fournissent des résultats de type `LOGICAL`. Ils sont donnés au tableau 3.2. Ce sont des opérateurs binaires entre variables numériques ou de type caractère.

Enfin, les opérateurs logiques permettent de combiner des expressions de type `LOGICAL` entre elles et fournissent des résultats de type `LOGICAL`. Ils sont

Table 3.1:

Opération	opérateur
Exponentiation	`**`
Multiplication	`*`
Division	`/`
Addition	`+`
Soustraction	`-`
Identité	`+`
Opposé	`-`

Table 3.2: Opérateurs relationnels

Opération	opérateurs
strictement plus petit	`.LT.` ou $<$
inférieur ou égal	`.LE.` ou $<=$
strictement supérieur	`.GT.` ou $>$
supérieur ou égal	`.GE.` ou $>=$
égal	`.EQ.` ou $==$
différent	`.NE.` ou $/=$

Table 3.3: Opérateurs logiques

Opération	opérateurs
négation	`.NOT.`
et logique	`.AND.`
ou logique	`.OR.`
équivalence logique	`.EQV.`
non équivalence logique	`.NEQV.`

repris au tableau 3.3.

Les opérateurs logiques sont évalués en dernier lieu, après les opérations relationnelles.

Chapter 4

Branchements et structures de contrôle

Les instructions contenues dans un programme Fortran sont exécutées de manière séquentielle. Afin de permettre des ordres d'exécution différents selon des cas à définir, Fortran 90 offre différents types de branchement et de structures de contrôle.

4.1 Branchement conditionnel

[étiquette:] `IF (`*expression logique*`) THEN`
  Bloc 1
[`ELSEIF (`*expression logique*`) THEN`]
  Bloc 2
...
[`ELSE`]
  Bloc n
`ENDIF` [étiquette]

On a l'option de mettre une étiquette à un bloc `IF`, ce qui permet de rendre le programme plus lisible, surtout lorsqu'un `IF` et son `ENDIF` sont éloignés l'un de l'autre.

Il existe une version courte, sans les alternatives `ELSEIF` et `ELSEIF` :

`IF (`*expression logique*`)` *instruction*

4.2 Choix multiple

[étiquette:] SELECT CASE (*var*)
  CASE (*valeur(s)1*)
    Bloc 1
  CASE (*valeur(s)2*)
    Bloc 2
  ...
  [CASE DEFAULT
    Bloc_par_défaut]
END SELECT [étiquette]

Bloc x est exécuté si la variable *var* possède la valeur *valeurx* ou se trouve dans la fourchette *valeursx* (donnée sous la forme (*valeurinf* : *valeursup*)). CASE DEFAULT permet d'exécuter un bloc d'instructions données lorsqu'aucun des cas explicitement inclus n'est applicable.

A noter encore que la variable *var* doit être de type INTEGER, LOGICAL ou CHARACTER, de manière à ne considérer que des tests permettant de vérifier des égalités réalisables de manière exacte.

4.3 La boucle DO

[étiquette:] DO [*contrôle_de_boucle*]
  Instructions
END DO [étiquette]

Pour des itérations, *contrôle_de_boucle* se met sous la forme

compteur = *compteur_ini*, *compteur_fin* [, *pas_compteur*]

Lorsque *pas_compteur* est omis, sa valeur est par défaut fixée à 1. L'itération démarre avec compteur = *compteur_ini*. Les instructions à l'intérieur de la boucle ne sont exécutées que si compteur est inférieur ou égal à *compteur_fin* (et donc, éventuellement, pas du tout). A la fin des instructions (au END DO), compteur est incrémenté de *pas_compteur*, et, si la valeur de compteur reste inférieure ou égale à *compteur_fin*, la boucle est ré-exécutée, sinon, le programme poursuit avec les instructions suivant END DO.

Une boucle avec un nombre indéterminé d'itérations peut se faire en utilisant un bloc *contrôle_de_boucle* vide. La sortie se fait alors à l'aide de l'instruction EXIT [étiquette]. Le déroulement de la boucle peut être altéré en forçant le passage suivant à l'aide de CYCLE [étiquette].

Enfin, une boucle DO WHILE est obtenue en utilisant pour *contrôle_de_boucle* l'expression WHILE (*expression_logique*)

Chapter 5

Fonctions mathématiques et opérations courantes

5.1 Fonctions intrinsèques

Presque toutes les fonctions mathématiques courantes sont disponibles sous Fortran 90 sous forme intrinsèque. Quelques fonctions des plus courantes sont reprises au tableau 5.1.

Table 5.1: Fonctions mathématiques courantes

Fonction mathématique	Fonction Fortran	Remarque
$\sin(x)$	`SIN(X)`	x en radians; résultat de même type que x
$\cos(x)$	`COS(X)`	x en radians; résultat de même type que x
$\tan(x)$	`TAN(X)`	x en radians; résultat de même type que x
$\text{asin}(x)$	`ASIN(X)`	résultat en radians et de même type que x
$\text{acos}(x)$	`ACOS(X)`	résultat en radians et de même type que x
$\text{atan}(x)$	`ATAN(X)`	résultat en radians et de même type que x
$\exp(x)$	`EXP(X)`	résultat de même type que x
$\ln(x)$	`LOG(X)`	résultat de même type que x
$\log_{10}(x)$	`LOG10(X)`	résultat de même type que x
$\sqrt{x}$	`SQRT(X)`	résultat de même type que x

5.2 Tableaux: propriétés et manipulation

5.3 Opérations diverses

Multiplication matricielle
- Maximum et minimum d'une liste
- Maximum et minimum d'un tableau
- Somme d'un tableau ou sous-tableau
- Longueur d'une chaîne de caractères
- Ceiling, Floor, nint, ...

Chapter 6

Opérations d'entrée-sortie

Instructions `OPEN`, `CLOSE`
- Mots-clé importants: `UNIT`, `FILE`, `STATUS`, `FORM`, `ACTION`, `IOSTAT`
- Instructions `PRINT`, `WRITE`, `READ`
- Instructions `INQUIRE`
- Le formattage des entrées-sorties. `FORMAT`
- `NAMELIST`

Chapter 7

Modules

Mise à disposition de paramètres, variables communes, procédures
`PRIVATE`, `PUBLIC`