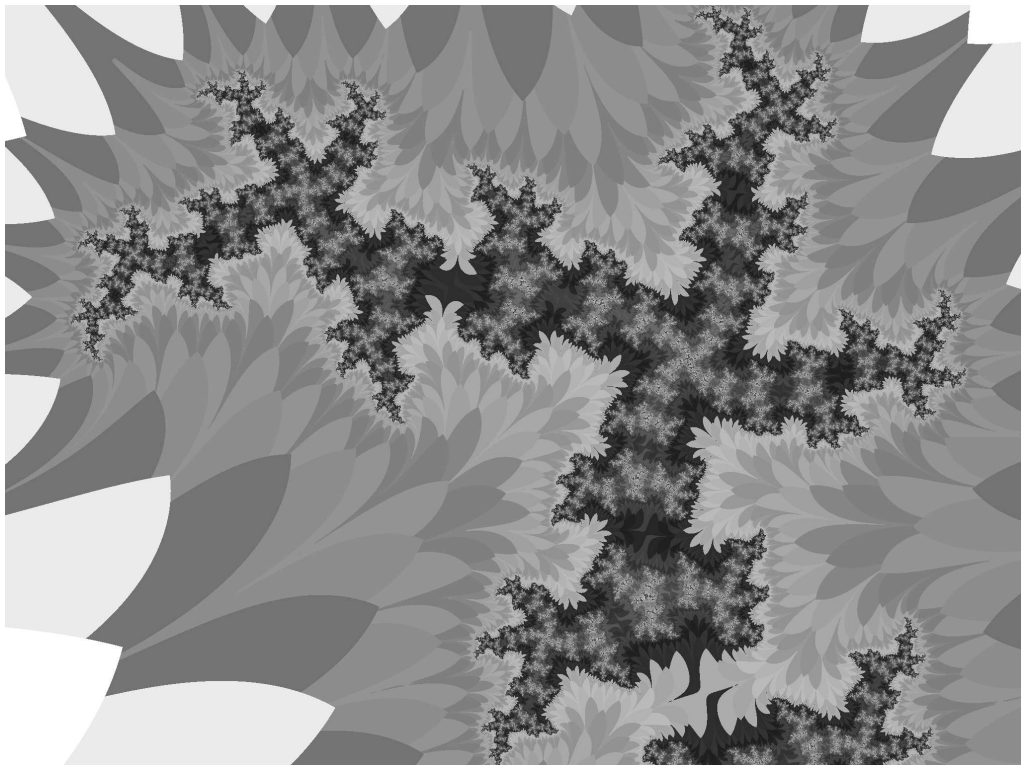




Fortran 77

Langage d'un autre age



MCours.com

Table des matières

| | | |
|----------|---|-----------|
| 1 | Introduction au FORTRAN | 1 |
| 1.1 | Histoire du FORTRAN | 1 |
| 1.2 | Conventions de base | 1 |
| 1.2.1 | Type de données | 1 |
| 1.2.2 | Variables | 2 |
| 1.2.3 | Structure d'un programme | 3 |
| 1.3 | Fonctions essentielles | 5 |
| 1.3.1 | Gestion des interactions | 5 |
| 1.3.2 | Opérateurs arithmétiques | 5 |
| 1.3.3 | Opérateurs de chaînes de caractères | 7 |
| 1.3.4 | Opérateurs logiques | 7 |
| 1.4 | Tableaux de données | 9 |
| 1.5 | Bibliothèque de fonctions | 9 |
| 2 | Structures de programmation | 11 |
| 2.1 | Structures fonctionnelles | 11 |
| 2.1.1 | Structures conditionnelles | 11 |
| 2.1.2 | Boucles conditionnelles | 12 |
| 2.1.3 | Boucle itérative | 13 |
| 2.1.4 | Boucle itérative implicite | 14 |
| 2.2 | Structures séquentielles | 14 |
| 2.2.1 | Fonctions | 14 |
| 2.2.2 | Blocs fonctionnels | 15 |
| 2.2.3 | Procédures | 17 |
| 3 | Compléments sur le FORTRAN | 21 |
| 3.1 | Gestion avancée des données | 21 |
| 3.1.1 | Types de données supplémentaires | 21 |
| 3.1.2 | Instruction de gestion des données | 21 |
| 3.2 | Gestion des entrées/sorties | 23 |
| 3.2.1 | Fichiers de données | 23 |
| 3.2.2 | Lecture et écriture dans un fichier | 24 |
| 3.2.3 | Formatage des données | 24 |
| 3.3 | Bibliothèque étendue de fonctions | 25 |
| 3.4 | La fonction <i>RANDOM</i> | 27 |
| 4 | Un exemple de programme complet | 29 |
| 4.1 | Solution d'un système linéaire | 29 |
| 4.2 | Pivot de Gauss | 29 |
| 4.3 | Exemple de programme FORTRAN | 30 |
| A | Références et licence | 35 |

Chapitre 1

Introduction au FORTRAN

Ce chapitre a pour but de présenter de manière relativement condensée le langage FORTRAN.

1.1 Histoire du FORTRAN

D'après Jean Sammet¹, le premier document faisant référence au langage FORTRAN date du 10 novembre 1954. FORTRAN vient, en fait, de *The IBM Mathematical Formula Translation System*. Il a été initialement conçu pour simplifier la programmation de calculs numériques sur les plateformes IBM 704.

La première version du FORTRAN n'est apparue qu'au début de l'année 1957 et même si les programmes obtenus à partir de code FORTRAN étaient plus lents que ceux obtenus à partir de codes en langage machine, le FORTRAN s'est imposé auprès de la communauté scientifique : il était bien plus simple à écrire. Très rapidement, il a été possible de réutiliser des codes FORTRAN sur d'autres plateformes que celles d'IBM.

Au début des années soixante, est apparue une myriade de compilateurs FORTRAN qui n'obéissaient pas exactement aux mêmes conventions de syntaxe. En 1966, il y eu une première tentative de normalisation du langage (travaux du Working Group X3.4.3 de l'American Standards Association) mais le groupe de travail n'a pas réussi à s'entendre sur un standard unique. C'est ainsi que sont nées deux normalisations distinctes : *FORTRAN* et *Basic FORTRAN*. En 1978, le même groupe de travail s'est de nouveau réuni et les spécifications du *FORTRAN 77* furent adoptées.

Depuis, le langage a encore évolué, deux autres standards sont nés, *FORTRAN 90* et *FORTRAN 95*, mais cela est une autre histoire...

1.2 Conventions de base

Comme tout langage, le FORTRAN utilise des règles pour fonctionner. Tout d'abord, nous verrons les règles de typage des données ainsi que des structures essentielles à l'implémentation d'algorithmes. Ensuite, nous verrons les conventions d'écriture d'un programme FORTRAN nécessaire à sa compilation et à son exécution.

1.2.1 Type de données

Le FORTRAN possède trois types de données : numériques, alphanumériques, et logiques. Chacune d'elles doit être utilisée avec discernement.

Données numériques

Les données numériques se subdivisent en *entiers*, *rationnels*, et *complexes*.

Le type INTEGER. Ce sont les nombres appartenant à \mathbb{Z} . Pour écrire des nombres entiers en FORTRAN, il suffit, simplement, d'entrer leurs valeurs (avec ou sans signe) sans toutefois utiliser de point décimal (les rationnels ne sont pas des entiers).

Exemple :

¹Sammet, J., *Programming Languages : History and Fundamentals* (Englewood Cliffs, NJ : Prentice-Hall, 1974)

```

5           -32          42
23422      0           -132323

```

Le type REAL. Ce sont, en fait, les nombres rationnels (appartenant à \mathbb{Q}). Ils peuvent être écrits de façon classique ou sous forme condensée avec une mantisse et un exposant².

Exemple :

```

3.1416      -1.          1.4E-18

```

Le type COMPLEX. Les nombres complexes sont aussi directement utilisables en FORTRAN (c'est à la base un langage à vocation scientifique). La partie réelle et la partie imaginaire sont considérées comme deux nombres rationnels assemblés entre parenthèses et séparés par une virgule.

Exemple :

```

(2, 3.21)   (3.4, -3)   (2, 3)

```

Données alphanumériques

Les chaînes de caractères, c'est-à-dire les mots, phrases, paragraphes ou même simples caractères, s'écrivent de deux façons différentes qui sont équivalentes.

Notation de Hollerith. C'est un lointain héritage des premiers compilateurs FORTRAN. Le nombre de caractères de la chaîne est précisé, suivi de la lettre « H », puis la chaîne elle-même est écrite.

Exemple :

```

10HNOIR DESIR

```

Ici, la chaîne comporte 10 caractères. L'espace au milieu est compris dans le comptage.

Notation entre quotes. La deuxième méthode est bien plus simple. Il suffit de mettre la chaîne entre quotes. Exemple :

```

'CLUB'      '666 667'    'BERTRAND CANTAT'

```

Données logiques

Ce sont des données booléennes (qui ne peuvent prendre que deux valeurs, soit Vrai soit Faux). Cela s'écrit :

```

.True.
.False.

```

1.2.2 Variables

Les variables sont des cases mémoire permettant de stocker des données. Les noms des variables obéissent à des règles strictes :

- Seules les lettres (de A à Z) et les chiffres (de 0 à 9) sont autorisés.
- Le premier caractère du nom doit être une lettre.
- La taille des noms de variables ne doit pas dépasser 6 caractères.

Variables typées

Pour chacun des types de données il existe un mot clé permettant de déclarer une variable du type désiré. Ainsi

```

INTEGER Liste-de-variables

```

permet de déclarer des variables de type entier³. De même

```

REAL Liste-de-variables

```

permet de déclarer des variables de type rationnel et

²c'est-à-dire en utilisant la notation scientifique mantisse $\times 10^{\text{exposant}}$

³Une liste de variables est un ou plusieurs noms de variables séparés par des virgules.

```
COMPLEX Liste-de-variables
```

pour les variables de type complexe.

```
LOGICAL Liste-de-variables
```

pour les variables de type booléen.

```
CHARACTER V1*N1,V2*N2, . . . ,VN*NN
```

déclare des variables de type chaîne de caractères. $N1, N2, \dots, NN$ représentent les nombres de caractères stockables dans chacune des variables. Si toutes les chaînes de caractères doivent avoir la même taille, il est aussi possible d'écrire :

```
CHARACTER*N Liste-de-variables
```

Déclaration implicite

D'autre part, il est possible de déclarer un ensemble de noms de variables d'un certain type. L'instruction `IMPLICIT` permet de déclarer que toutes les variables dont le nom commence par une certaine lettre sont d'un même type. Sa syntaxe est :

```
IMPLICIT Type (L1-L2)
```

$L1$ et $L2$ sont les deux lettres extrêmes ($L2$ et le tiret sont facultatifs si l'on ne veut déclarer d'un même type que des variables commençant par une seule même lettre) et `Type`, le type commun.

Variables non déclarées

En FORTRAN, si une variable est utilisée sans avoir été déclarée, son type dépend de la première lettre de son nom. Si son nom commence par I, J, K, L, M ou N c'est un entier. Sinon c'est un rationnel. Il est toutefois **fortement** recommandé de toujours déclarer les variables de manière explicite.

1.2.3 Structure d'un programme

La structure d'un programme en FORTRAN (comme pour tout autre langage de programmation) obéit à certaines règles.

Mise en page d'un programme

Une ligne en FORTRAN ne peut contenir qu'au maximum 80 caractères. De plus, ces 80 caractères se décomposent en quatre champs différents :

- Du premier au cinquième se trouve le champ dit de référence. Il permet d'attribuer un numéro (de 1 à 99999) à la ligne pour pouvoir y accéder directement via des fonctions de déplacement dans le programme. Toutefois, c'est aussi dans ce champ que l'on peut déclarer qu'une ligne n'est pas à prendre en compte mais est, en fait, un commentaire, en utilisant comme premier caractère un « C » ou un « * ».
- Le sixième caractère indique que la ligne actuelle n'est que la suite de la ligne précédente. C'est très utile pour pouvoir entrer des lignes de calculs ou de programmation comportant plus de 66 caractères. En effet, tout caractère (excepté « 0 ») permet de prolonger la ligne précédente. Par convention, nous utiliserons dans ce manuel le caractère « + ».
- Le troisième champ (c'est-à-dire du caractère 7 au caractère 72) comprend véritablement les commandes FORTRAN.
- Enfin, le dernier champ (de 73 à 80) n'est pas interprété par le compilateur FORTRAN 77. Il permet de commenter une ligne de commande.

La commande *PROGRAM*

Cette commande n'est pas obligatoire en FORTRAN 77 mais elle permet de nommer un programme. Si elle existe, ce doit être la première ligne d'un programme. Sa syntaxe est :

```
PROGRAM Name
```

Il est recommandé de commencer tout programme par cette ligne. C'est une aide à la programmation.

La commande *END*

C'est la dernière commande de tout programme FORTRAN. Celle-ci, par contre, est obligatoire. Sa syntaxe est :

```
END
```

Toute ligne écrite au delà n'est pas prise en compte ou bien, celle-ci plante le compilateur (cela dépend de l'implémentation de ce dernier).

La commande *STOP*

Elle permet de sortir d'un programme sans toutefois avoir atteint la dernière ligne du programme. Sa syntaxe est :

```
STOP
```

Elle est strictement équivalente à la commande *END* mais plus souple d'utilisation puisqu'elle permet d'avoir plusieurs points de sortie dans un même programme et qu'elle permet de renvoyer un code de référence.

L'instruction *DATA*

Elle permet de stocker des valeurs dans des variables sans faire d'opération d'affectation. Elle n'est pas exécutée par le programme mais directement interprétée par le compilateur qui, au lieu d'initialiser des variables à zéro, les initialisera à la valeur souhaitée. Sa syntaxe est :

```
DATA Liste_de_variables/Liste_de_valeur/
```

La syntaxe de la liste de valeur est la suivante :

- Chacune des valeurs est séparée par une virgule.
- Si N variables doivent être initialisées à la même valeur, il est possible de ne pas répéter N fois cette valeur et d'écrire Valeur*N.

Exemple :

```
REAL A,B,C,D
INTEGER I,J,K
DATA A,B,C,I/1.0,2.0,3.0,-1/
DATA J,K/-6*2/
```

La commande *GO TO*

Elle permet de se déplacer dans un programme sans exécuter d'instruction. Sa syntaxe est :

```
GO TO N
```

ou N indique le numéro de ligne où le programme doit reprendre son exécution.

La commande *CONTINUE*

Cette commande ne fait rien. Elle peut être utile pour remplir des lignes de programme dont le principal intérêt est leur numéro de référence.

Structure d'un bloc

Comme cela a été vu précédemment, tout programme commence par la commande *PROGRAM* et se termine par la commande *END*. Toutefois, entre ces deux commandes, les instructions se distinguent en deux parties :

- La partie déclarative contient les instructions relatives aux déclarations de variables, de constantes, de paramètres et de formats. La plupart de ces instructions seront développées dans les chapitres qui vont suivre.
- La partie fonctionnelle contient véritablement l'algorithme du programme.

Il est bien évident que si l'on distingue deux parties, c'est que les déclarations en tout genre ne doivent pas apparaître entre deux commandes et vice-versa, c'est-à-dire que la partie déclarative commence tout programme et est suivie de la partie fonctionnelle.

1.3 Fonctions essentielles

Maintenant que nous savons écrire à la mode FORTRAN, il ne nous manque que les ingrédients pour commencer notre cuisine.

1.3.1 Gestion des interactions

Dans tout programme, il est nécessaire d'avoir des interactions entre ce dernier et l'utilisateur. C'est-à-dire, permettre à l'utilisateur d'entrer des données et au programme d'afficher des résultats.

La commande *READ*

Elle permet de lire des données entrées via un périphérique (essentiellement le clavier ou un fichier). Sa syntaxe est :

```
READ*, Liste_de_variables
READ(N,*) Liste_de_variables
```

Où N est l'identificateur d'un flux de données⁴. Pour le clavier, c'est généralement 5. Quelle que soit la forme utilisée, lors de l'exécution du programme, il faut, bien sûr, respecter le type des données.

La commande *PRINT*

Elle permet d'afficher des informations ou des variables à l'écran. Sa syntaxe est :

```
PRINT*, Liste_a_afficher
```

On peut donc écrire :

```
REAL A,B
PRINT*, 'Entrer deux nombres'
READ*, A,B
PRINT*, 'Le resultat de',A,'+',B,' est :',A+B
```

La commande *WRITE*

Cette commande est équivalente à *PRINT* mais sa syntaxe est légèrement différente.

```
WRITE(N,*) Liste_a_afficher
```

Comme pour la deuxième forme de la commande *READ*, N est l'identificateur d'un flux de données. Pour l'écran, c'est généralement 6.

1.3.2 Opérateurs arithmétiques

Le FORTRAN possède six opérateurs arithmétiques dont il faut connaître les différentes priorités.

L'addition

C'est l'opérateur « + »

```
NOMBRE1+NOMBRE2
```

La soustraction

C'est l'opérateur « - »

```
NOMBRE1 - NOMBRE2
```

⁴Pour plus de détails, se reporter au chapitre 3.2.1

La multiplication

C'est l'opérateur « * »

NOMBRE1*NOMBRE2

La division

C'est l'opérateur « / »

NOMBRE1/NOMBRE2

Si le numérateur ou le dénominateur est un rationnel, le résultat est rationnel. Si ce sont des entiers, c'est en fait une division euclidienne et le résultat un entier.

La puissance

C'est l'opérateur « ** »

NOMBRE1**NOMBRE2

Il permet de calculer x^y . Comme les deux nombres peuvent être des rationnels, il est possible de calculer des exposants fractionnaires.

La négation

C'est l'opérateur unaire « - »

-NOMBRE1
(-NOMBRE1)

L'affectation

Ce n'est pas un opérateur arithmétique mais c'est un opérateur pour le langage FORTRAN. On peut donc écrire :

VAR = NOMBRE
VAR1 = VAR2

Il faut, bien sûr, que les deux membres de l'affectation soient de types compatibles.

Hiérarchie des opérateurs

Chacun des opérateurs a une priorité différente permettant d'évaluer une expression de façon unique. Ainsi :

- L'exposant a la plus haute priorité.
- La multiplication et la division ont une priorité intermédiaire.
- L'addition, la soustraction et la négation sont de plus basse priorité.

Pour clarifier des situations ambiguës, les parenthèses permettent de regrouper des calculs intermédiaires.

Pour traduire en FORTRAN l'expression mathématique

$$x^2 - \frac{x + x^3}{x - \frac{1}{x}}$$

on peut écrire :

X**2+(X+X**3)/(X-1/X)
X*(X+(1+X**2)/(X-1/X)
X**2*(1+(1+X**2)/(X**2-1))

Toutefois, comme les commandes sont interprétées de gauche à droite, deux opérateurs de même priorité travaillant sur des données de types différents ne vont pas donner le même résultat suivant l'ordre des données elles-mêmes.

Exemple :

```

M = 6
N = 5
A = 2.5
B = M/N*A
C = A*M/N

```

B va valoir 2.5 et C 3.0. Surprenant, non ?

1.3.3 Opérateurs de chaînes de caractères

Il existe aussi des opérateurs permettant d'effectuer les opérations standards sur les chaînes de caractères, c'est-à-dire la concaténation et l'extraction de chaînes de caractères.

La concaténation

C'est l'opérateur « // »

```
Chaine1//Chaine2
```

Si l'on veut stocker le résultat dans une troisième chaîne, il faut, bien sûr, lui prévoir une taille suffisamment grande pour recevoir les deux autres chaînes.

L'extraction

C'est un opérateur unaire

```
Chaine(I:J)
```

ou I et J sont deux entiers inférieurs ou égaux à la taille de la chaîne de caractères et I est inférieur ou égal à J. Ils représentent la position du premier et du dernier caractère à extraire. L'un des deux indices peut être négligé pour extraire simplement les J premiers caractères ou les J derniers.

Attention, une opération de type

$$b = a + b$$

peut s'avérer catastrophique. En effet, le compilateur FORTRAN effectue les opérations sur les chaînes, caractère par caractère. Et un programme tel que :

```

PROGRAM TEST
CHARACTER CH*9, CH1*2
CH = ' revoir'
CH1 = 'au'
CH = CH1 // CH(1:7)
PRINT*, CH
STOP
END

```

affichera

```
auauauaua
```

1.3.4 Opérateurs logiques

Ces opérateurs peuvent être regroupés en deux catégories. Certains permettent de comparer des données de même type, d'autres permettent d'effectuer des opérations booléennes sur des données logiques.

| Opérateur | Syntaxe en FORTRAN | Sens |
|-----------|--------------------|---------------------------------|
| .EQ. | A.EQ.B | A est égal à B |
| .NE. | A.NE.B | A est différent de B |
| .LT. | A.LT.B | A est strictement inférieur à B |
| .GT. | A.GT.B | A est strictement supérieur à B |
| .LE. | A.LE.B | A est inférieur ou égal à B |
| .GE. | A.GE.B | A est supérieur ou égal à B |

TAB. 1.1 – Table des opérateurs d'ordre.

Relation d'ordre

Ces opérateurs comparent deux données de type INTERGER, REAL ou CHARACTER en fonction d'un certain critère pour déterminer si le résultat est vrai ou faux. Leur syntaxe est :

```
Operande1 OPERATEUR Operande2
```

La liste des opérateurs se trouve dans le tableau 1.1.

Ainsi

```
42 .LE. 23.0
```

est faux puisque 43 est supérieur à 23. Notons que le FORTRAN a converti l'entier 43 en rationnel pour effectuer la comparaison. De la même façon, il est possible de comparer des chaînes de caractères.

```
'ABCE' .GT. 'ABD'
```

Cette commande renvoie faux car le code ASCII de « C » est plus petit que celui de « D » .

Opérateurs booléens

Ils permettent d'effectuer les trois opérations booléennes standards.

Et logique Sa syntaxe est :

```
A .AND. B
```

Son résultat est vrai si A et B sont vrais.

Ou logique Sa syntaxe est :

```
A .OR. B
```

Son résultat est vrai si A ou B sont vrais.

Equivalent logique Sa syntaxe est :

```
A .EQV. B
```

Son résultat est vrai si A et B sont de même valeur (vrais ou faux en même temps).

Ou exclusif logique Sa syntaxe est :

```
A .NEQV. B
```

Son résultat est vrai si seul A ou B est vrai.

Non logique Sa syntaxe est :

```
.NOT. A
```

Son résultat est vrai si A est faux.

Il est, bien sûr, possible d'écrire des expressions conditionnelles complexes en utilisant tous ces opérateurs logiques ainsi que les opérateurs arithmétiques et de chaîne de caractères. Il est alors prudent de mettre entre parenthèses les différentes expressions car les opérateurs logiques ont les plus faibles priorités.

1.4 Tableaux de données

Les tableaux de données permettent de stocker des vecteurs, des matrices ou même des tenseurs (généralisation des matrices en trois dimensions). Ce n'est pas un type de données à part entière mais plutôt une extension des types standards.

Il existe deux syntaxes possibles :

```
Type Nom_de_variable
DIMENSION Nom_de_variable(N1,N2,N3)
```

ou

```
Type Nom_de_variable(N1,N2,N3)
```

De ces deux façons sont définis des tableaux contenant tous des données d'un même type. $N1, N2, N3$ représentent les tailles de chacune des trois dimensions ($N2$ et $N3$ sont facultatives si l'on ne veut définir qu'un tableau à une dimension).

Exemple :

```
INTEGER A
DIMENSION A(10)
REAL B(10,5,3)
COMPLEX AA(10,2)
CHARACTER C(10)*4
```

A est un vecteur contenant 10 entiers, B un tenseur ($10 \times 5 \times 3$) de rationnels, AA une matrice (10×2) de nombres complexes et C un tableau de chaîne à 10 entrées de 4 caractères.

L'utilisation des tableaux est la suivante : si l'on veut, par exemple, récupérer la 3^e composante de A, il suffit d'écrire `A(3)`, et ainsi de suite.

Il est aussi possible de définir des tableaux ne commençant pas par l'indice 1. Ainsi :

```
Type Nom_de_variable
DIMENSION Nom_de_variable(P1:D1,P2:D2,P3:D3)
```

permet de définir des tableaux dont les indices sont compris entre P1 (respectivement P2 et P3) et D1 (respectivement D2 et D3).

Exemple :

```
INTERGER RETARD(-5:0)
```

RETARD est un tableau dont le premier indice est -5 et le dernier 0.

1.5 Bibliothèque de fonctions

Heureusement pour les programmeurs FORTRAN, il existe des bibliothèques de fonctions (appelées couramment *libraries*) pour calculer, par exemple, des racines carrées, déterminer la valeur absolue d'un nombre... Pour exploiter ces fonctions, il suffit de taper le nom de celle que l'on veut utiliser. En voici quelques unes.

```
R = SQRT(R1)
```

Calcule la racine carrée d'un rationnel, le résultat est aussi un rationnel.

```
I = INT(R)
```

Arrondit par défaut un rationnel à l'entier le plus proche.

```
R = ABS(R1)
```

Calcule la valeur absolue d'un rationnel, le résultat est aussi un rationnel.

```
I = MAX0(I1,I2, . . . ,IK)
```

Détermine le maximum des entiers $I_1, I_2 \dots I_K$.

$$I = \text{MIN0}(I_1, I_2, \dots, I_K)$$

Détermine le minimum des entiers $I_1, I_2 \dots I_K$.

$$R = \text{FLOAT}(I)$$

Génère un rationnel à partir de l'entier I .

$$I = \text{MOD}(I_1, I_2)$$

Calcule le reste de la division euclidienne de I_1 par I_2 .

$$R = \text{EXP}(R_1)$$

Calcule l'exponentielle d'un rationnel.

$$I = \text{LEN}(CH)$$

Détermine la taille d'une chaîne de caractères.

$$I = \text{ICHAR}(C)$$

Retourne le code ASCII d'un caractère.

$$C = \text{CHAR}(I)$$

Retourne le caractère correspondant à un code ASCII.

$$I = \text{INDEX}(CH_1, CH_2)$$

Recherche une sous-chaîne dans une autre chaîne de caractères. Si celle-ci existe, la fonction renvoie la position du caractère le plus à gauche.

$$R = \text{RANDOM}(1)$$

Calcule un nombre aléatoire dans $[0, 1[$. Le paramètre de cette fonction, bien qu'obligatoire, n'est absolument pas pris en compte.

Chapitre 2

Structures de programmation

Au cours de ce chapitre, nous verrons les différentes structures de programmation qui permettent à un ordinateur d'être autre chose qu'une simple machine à calculer.

2.1 Structures fonctionnelles

Ces structures permettent, à l'intérieur d'un programme, de regrouper des commandes pour pouvoir les exécuter à sa convenance.

2.1.1 Structures conditionnelles

Ces structures donnent le choix entre deux ensembles de commandes. Il en existe quatre qui diffèrent essentiellement au niveau de la complexité des commandes à effectuer.

Structure *IF* logique

Cette structure est utile lorsqu'une seule commande dépend d'une condition. Sa syntaxe est :

```
IF (Condition) Commande
```

La commande n'est exécutée que si la condition est vraie. La condition est une valeur ou une variable logique.

Structure *IF-THEN*

Celle-ci est à utiliser lorsqu'on a plusieurs commandes à effectuer conditionnellement. Sa syntaxe est :

```
IF (Condition) THEN  
  Commande 1  
  Commande 2  
  .  
  .  
  Commande N  
ENDIF
```

Les N commandes sont exécutées si la condition est vraie.

Structure *IF-THEN-ELSE*

Cette troisième permet le choix d'un ensemble de commandes ou d'un autre. Sa syntaxe est :

```
IF (Condition) THEN  
  Ensemble_de_commandes 1  
ELSE
```

```

    Ensemble_de_commandes 2
ENDIF

```

Le premier ensemble de commandes est exécuté si la condition est vraie, sinon le second ensemble est exécuté.

Structure *IF-THEN-ELSEIF-ELSE*

Cette dernière est la plus complexe des structures conditionnelles. Elle permet d'exécuter un certain ensemble de commandes en fonction d'une certaine condition. Sa syntaxe est :

```

IF (Condition 1) THEN
    Ensemble_de_commandes 1
ELSEIF (Condition 2) THEN
    Ensemble_de_commandes 2
ELSEIF (Condition 3) THEN
    Ensemble_de_commandes 3
.
.
ELSEIF (Condition N-1) THEN
    Ensemble_de_commandes N-1
ELSE
    Ensemble_de_commandes N
ENDIF

```

Le premier ensemble de commandes est exécuté si la première condition est vraie, sinon, c'est le deuxième ensemble qui sera exécuté si la deuxième condition est vraie, et ainsi de suite. Si aucune condition n'est vraie, alors le dernier ensemble de commandes sera exécuté. Ce dernier ensemble ainsi que la commande *ELSE* est facultatif.

2.1.2 Boucles conditionnelles

Ces deux structures permettent de répéter un ensemble d'instructions en fonction d'une condition. Elles sont strictement équivalentes. Toutefois, il est préférable d'utiliser la seconde qui est bien plus lisible.

Boucle *IF-GO TO*

Sa syntaxe est :

```

10  IF (.NOT. Condition) GO TO 20
    Ensemble_de_commandes
GO TO 10
20  Commande_quelconque

```

L'ensemble de commandes est exécuté tant que la condition est vraie.

Boucle *WHILE-DO*

Cette structure permet de réécrire la structure précédente sans utiliser de référence. Sa syntaxe est :

```

WHILE (Condition) DO
    Ensemble_de_commandes
ENDWHILE

```

De plus, il n'est pas nécessaire d'avoir une commande juste après cette structure.

Un théorème de mathématiques a démontré l'équivalence de ces deux structures, alors **n'utilisez** de boucles de type *IF-GO TO* (faciles à boguer) que si le compilateur FORTRAN présent sur la station ne reconnaît pas les boucles *WHILE-DO*.

Calcul de Fibonacci

Voici un petit exemple inspiré par la suite de Fibonacci.

```

PROGRAM LIMITE
REAL U1, U2, U3, E
INTEGER I, R
PRINT*, 'Limite de la suite U(n+1) = U(n) - U(n-1)'
PRINT*, 'U(1) et U(2) ?'
READ*, U1, U2
PRINT*, 'Rang maximum'
READ*, R
PRINT*, 'Precision'
READ*, E
I = 3
WHILE ((I .LE. R) .AND. (ABS(U2-U1) .GT. E)) DO
  U3 = (U2-U1)/(U2+U1)
  U1 = U2
  U2 = U3
  I = I + 1
ENDWHILE
IF (ABS(U2-U1) .LE. E) THEN
  PRINT*, 'U(inf) = ', U2
ELSE
  PRINT*, 'Desole'
ENDIF
STOP
END

```

2.1.3 Boucle itérative

Cette structure permet de répéter un certain nombre de fois un ensemble de commandes. Sa forme générique est :

```

      DO R Indice = Exp1, Exp2, Exp3
        Ensemble_de_commandes
R     CONTINUE

```

où

- R est le numéro de la ligne CONTINUE (dernière ligne de l'ensemble) associé au DO.
- Indice est une variable de type entier.
- Exp1 est une expression représentant la première valeur de l'indice.
- Exp2 est une expression représentant la dernière valeur de l'indice (atteinte ou pas).
- Exp3 est une expression représentant le pas d'incrément. Par défaut, le pas est de 1.

Il existe une seconde forme de boucle itérative, plus simple à utiliser. Elle s'écrit :

```

      DO Indice = Exp1, Exp2, Exp3
        Ensemble_de_commandes
      END DO

```

Il faut noter que chaque boucle itérative se termine par un END DO qui lui est propre.

Gaus

Calculons, par exemple, la somme des n premiers entiers.

```

PROGRAM SOMME
INTEGER I,S,N
PRINT*, 'Calcul de la somme des N premiers entiers'

```

```

READ*, N
S = 0
DO 10 I = 1, N
  S = S + I
10 CONTINUE
PRINT*, 'Somme = ', S
STOP
END

```

2.1.4 Boucle itérative implicite

Pour certaines commandes (comme READ, WRITE, PRINT et DATA) il est possible de paramétrer en une seule ligne plusieurs boucles imbriquées en utilisant la syntaxe :

```
READ*, ((A(I,J,K), K=K1,K2,K3), J=J1,J2,J3), I=I1,I2,I3)
```

ou (K1, K2, K3), (J1, J2, J3) et (I1, I2, I3) décrivent chacune des boucles.

2.2 Structures séquentielles

Ces structures permettent de découper un programme en diverses parties. Ainsi, il est inutile de dupliquer des morceaux de codes qui interviennent plusieurs fois. De plus, il est plus facile de coder correctement plusieurs petits blocs d'une dizaine de lignes qu'un seul programme. Il est **fortement** conseillé d'utiliser ce genre de structures pour clarifier ses programmes, pour son confort personnel mais aussi pour celui de ceux qui le reliront ensuite.

Le nom de ces structures suit la même syntaxe que les noms de variables en FORTRAN. C'est-à-dire, pas plus de six caractères et le premier caractère doit être une lettre.

2.2.1 Fonctions

Ce sont des fonctions au sens mathématique du terme. C'est-à-dire,

$$\begin{array}{ccc} \mathbb{E} & \longrightarrow & \mathbb{K} \\ (x_1, x_2 \dots x_n) & \longrightarrow & y = f(x_1, x_2 \dots x_n) \end{array}$$

Elles permettent de définir des formules mathématiques pouvant être utilisées plusieurs fois. Leur syntaxe est :

```
Nom(Arguments) = Definition
```

Où Nom est le nom de la fonction, Arguments le ou les arguments de la fonction séparés par une virgule (leur type est absolument quelconque) et Definition la définition de la fonction. Comme la valeur que doit renvoyer la fonction est typée, il ne faut pas oublier de typer le nom de la fonction.

Exemple :

```

PROGRAM PARABOLE
REAL X,A,B,C
C
C Typage de la fonction
C
REAL F
CHARACTER TEST
C
C Definition de la fonction
C
F(X)=A*X*X+B*X+C
PRINT*, 'Calcul de f(x)=a*x^2+b*x+c'
PRINT*, 'Entrez a b c'

```

```

    READ*, A,B,C
10  PRINT*, 'Valeur de x'
    READ*, X
    PRINT*, 'f(x) = ', F(X)
    PRINT*, 'Une autre valeur (o/n)'
    READ*, TEST
    IF (TEST .EQ. 'o') GO TO 10
    STOP
    END

```

2.2.2 Blocs fonctionnels

Ce sont des ensembles de commandes qui s'appliquent à un certain nombre d'arguments et qui renvoient une valeur. Ils commencent par la commande `FUNCTION`, finissent par `END` et doivent se trouver placés après le `END` du programme.

Il existe deux syntaxes différentes pour déclarer un bloc fonctionnel.

```

FUNCTION Nom(Par1,Par2, . . . ParN)
Type Nom
Type1 Par1
Type2 Par2
.
.
Ensemble_de_commandes
.
.
END

```

Ou

```

Type FUNCTION Nom(Par1,Par2, . . . ParN)
Type1 Par1
Type2 Par2
.
.
Ensemble_de_commandes
.
.
END

```

La seconde forme est un peu plus simple à lire.

Le ou les paramètres (il en faut au minimum 1 et au maximum 63) doivent être définis dans le corps de la fonction. De plus, il faut qu'une valeur soit affectée à la variable du même nom que la fonction (appelée variable de retour), avant la commande `END`, pour être renvoyée par la fonction.

```
Nom = Valeur
```

Dans le programme, ces blocs fonctionnels s'utilisent comme les autres fonctions définies dans les bibliothèques. C'est-à-dire :

```
Result = Nom(Arg1,Arg2, . . . ArgN)
```

De la même manière que dans un programme, il est possible de définir des variables. Sauf exception¹, ces variables ne sont utilisables que dans le bloc fonctionnel lui-même.

Commande *RETURN*

Cette commande est un pendant de la commande `STOP` des programmes. Elle permet de sortir d'un bloc fonctionnel sans aller jusqu'à la commande `END`. Il faut toutefois que la variable de retour ait été remplie.

¹Pour plus de détails, se reporter au chapitre 3.1.2

Exemple

```

PROGRAM NOTE
C
C Predeclaration du type de la valeur de retour
C
REAL MOY
REAL PROJ, TP, TOT, FINAL
INTEGER NB, PAR
CHARACTER C

NB = 0
TOT = 0

PRINT*, 'Calcul des moyennes du module de Fortran'
10 PRINT*, ''
PRINT*, 'Note de projet '
READ*, PROJ
PRINT*, 'Participation au cours et au TP (0-5)'
READ*, PAR
PRINT*, 'Note de TP'
TP = MOY(PAR)
FINAL = (TP+PROJ)/2
PRINT*, 'Moyenne = ', FINAL
NB = NB+1
TOT = TOT+FINAL
PRINT*, 'Autre eleve (o/n)'
READ*, C
IF (C .EQ. 'o') GO TO 10
PRINT*, 'Moyenne : ', TOT/NB, ' sur ', NB, ' etudiants'
STOP
END

C
C Definition de la fonction
C
REAL FUNCTION MOY(P)
C
C Declaration des parametres
C
INTEGER P
C
C Variable propre a la fonction
C
INTEGER NBTP
REAL TP

MOY = 0
NBTP = 1

PRINT*, 'Note TP ', NBTP
READ*, TP

100 MOY = MOY+TP
NBTP = NBTP+1
PRINT*, 'Note TP ', NBTP, ' (-1 pour finir)'
READ*, TP

```

```

    IF (TP .NE. -1) GO TO 100

    MOY = MOY/(NBTP-1)+P

    RETURN
END

```

Le passage des arguments à une fonction est effectué *par variable si possible*. C'est-à-dire que si l'un des arguments est une variable et que la valeur de ce paramètre est changée au cours de l'exécution de la fonction, la variable d'appel est aussi changée en sortie de la fonction.

Exemple :

```

PROGRAM TRANSFERT
C
C Test du passage de parametre a une fonction
C
    INTEGER I,J,R
    INTEGER CHANGE
    PRINT*, 'Permutation de trois entiers'
    PRINT*, 'Entrez deux entiers'
    READ*, I,J
    PRINT*, 'Execution de CHANGE(I,J, I+J)'
    R = CHANGE(I,J, I+J)
    PRINT*, 'I = ',I
    PRINT*, 'J = ',J
    STOP
END

C
C Fonction de transfert
C
    INTEGER FUNCTION CHANGE(A,B,C)
    INTEGER A,B,C,T
    T = C
    C = A
    A = B
    B = T
    CHANGE = 0
    RETURN
END

```

Lors de l'appel de la fonction CHANGE, seuls les deux premiers arguments sont des variables, le troisième est une simple valeur. En sortie de la fonction, I et J ont changé et le programme continue de fonctionner même s'il n'a pas réussi à ré-affecter C.

2.2.3 Procédures

Les procédures sont très proches des blocs fonctionnels. Ce sont des ensembles de commandes s'appliquant à un certain nombre d'arguments (au maximum 63) qui doivent être définis après le END du programme. Leur syntaxe est :

```

SUBROUTINE Nom(Par1,Par2, . . . ParN)
Type1 Par1
Type2 Par2
.
.
Ensemble_de_commandes
.

```

```
.
END
```

La commande RETURN permet de sortir d'une procédure de la même façon que dans un bloc fonctionnel. Les variables définies à l'intérieur d'une procédure ne sont utilisables que dans la procédure.

Commande CALL

Cette commande permet d'appeler une procédure. Sa syntaxe est :

```
CALL Nom(Arg1,Arg2, . . . ArgN)
```

Exemple

Cet exemple, un peu complexe, exploite non seulement les structures procédurales mais aussi les structures de boucles en cascades, des boucles implicites et une structure conditionnelle complexe.

```
PROGRAM MULT_MAT
C
C Programme de multiplication matricielle (3x3)
C
REAL A(3,3),B(3,3)
CHARACTER C
C
C Initialisation
C
DATA A,B/18*0.0/
C
C Corps du programme principal
C
PRINT*, 'Programme de multiplication matricielle'
10 PRINT*, ' '
PRINT*, 'Voulez-vous : '
PRINT*, '(1)- Entrer A'
PRINT*, '(2)- Entrer B'
PRINT*, '(3)- Afficher A'
PRINT*, '(4)- Afficher B'
PRINT*, '(5)- Calculer A*B'
PRINT*, '(6)- Sortir'
READ*, C
IF (C .EQ. '1') THEN
PRINT*, ' -> A '
CALL LIRE(A)
ELSEIF (C .EQ. '2') THEN
PRINT*, ' -> B '
CALL LIRE(B)
ELSEIF (C .EQ. '3') THEN
PRINT*, ' A ='
CALL ECRIRE(A)
ELSEIF (C .EQ. '4') THEN
PRINT*, ' B ='
CALL ECRIRE(B)
ELSEIF (C .EQ. '5') THEN
PRINT*, ' A*B -> B'
CALL MULTAB(A,B)
ELSEIF (C .EQ. '6') THEN
STOP
```

```

ELSE
    PRINT*, 'Ordre non compris'
ENDIF
GO TO 10
END

C
C Procédure de lecture
C
SUBROUTINE LIRE(D)
    REAL D(3,3)
    INTEGER I,J
    PRINT*, ' Entrez la matrice par ligne'
    READ*, ((D(I,J), J=1,3), I=1,3)
    RETURN
END

C
C Procédure d'affichage
C
SUBROUTINE ECRIRE(D)
    REAL D(3,3)
    INTEGER I,J
    DO I=1,3
        PRINT*, (D(I,J), J=1,3)
    END DO
    RETURN
END

C
C Procédure de multiplication
C
SUBROUTINE MULTAB(D,E)
    REAL D(3,3),E(3,3)
    REAL F(3,3)
    INTEGER I,J, K
    DO I=1,3
        DO J=1,3
            F(I,J) = 0
            DO K=1,3
                F(I,J) = F(I,J)+D(I,K)*E(K,J)
            END DO
        END DO
    END DO
    CALL TRANS(F,E)
    RETURN
END

C
C Procédure de transfert
C
SUBROUTINE TRANS(D,E)
    REAL D(3,3),E(3,3)
    INTEGER I,J
    DO I=1,3
        DO J=1,3
            E(I,J) = D(I,J)
        END DO
    END DO
    RETURN

```

END

Chapitre 3

Compléments sur le FORTRAN

Ce chapitre traite d'une multitude de petits points. Bien que secondaires pour la compréhension du langage FORTRAN, ils en font toute sa richesse.

3.1 Gestion avancée des données

Pour le bon fonctionnement des programmes, il est utile de connaître la précision de calcul des différents types de données ainsi que quelques commandes qui simplifient les échanges de données.

3.1.1 Types de données supplémentaires

Les types de données détaillés dans le chapitre 1.2.1 sont les types standards du FORTRAN. Il est en fait possible de préciser la taille en octets des variables entières, rationnelles, complexes ou logiques. Le tableau 3.1 présente les types dérivés. Suivant la place mémoire accordée, les valeurs prises par ces variables

| Type | Taille mémoire en octets |
|------------|--------------------------|
| INTEGER | (2) ou 4 |
| INTEGER*2 | 2 |
| INTEGER*4 | 4 |
| REAL | 4 |
| REAL*4 | 4 |
| REAL*8 | 8 |
| COMPLEX | 8 |
| COMPLEX*8 | 8 |
| COMPLEX*16 | 16 |
| LOGICAL | (2) ou 4 |
| LOGICAL*2 | 2 |
| LOGICAL*4 | 4 |

TAB. 3.1 – Mémoire et type.

diffèrent. Le tableau 3.2 présente de façon sommaire, les valeurs maximales et minimales de chacun des types.

3.1.2 Instruction de gestion des données

Ces instructions facilitent les échanges de données entre les blocs fonctionnels, les procédures et le programme principal.

| Type | Précision |
|-----------|--|
| INTEGER*2 | $+2^{15} - 1 \dots 0 \dots -2^{15} - 1$ |
| INTEGER*4 | $+2^{31} - 1 \dots 0 \dots -2^{31} - 1$ |
| REAL*4 | $+3,37 * 10^{+38}$ à $8,43 * 10^{-37} \dots 0 \dots -8,43 * 10^{-37}$ à $-3,37 * 10^{+38}$ |
| REAL*8 | $+1,67 * 10^{+308}$ à $4,19 * 10^{-307} \dots 0 \dots -4,19 * 10^{-307}$ à $-1,67 * 10^{+308}$ |
| LOGICAL*2 | -1 ou 0 |
| LOGICAL*4 | -1 ou 0 |

TAB. 3.2 – Précision et type.

Instruction COMMON

Cette instruction permet de partager une (ou plusieurs) variable(s) entre plusieurs blocs de programmation. Sa syntaxe est :

```
COMMON /Nom/ Liste_de_variable
```

Nom définit l'identificateur de l'ensemble des variables à partager. Il suit les règles des noms de variables.

Exemple :

```
PROGRAM PARTAGE
C
C Partage de donnees
C
REAL A(3),B
COMMON /DONNEES/ A
PRINT*, 'Calcul de (x+y)/z'
PRINT*, 'Entrez x, y, z'
READ*, A
CALL CALCUL(B)
PRINT*, B
STOP
END

C
C Procedure de calcul
C
SUBROUTINE CALCUL(R)
REAL R
REAL A1,A2,A3
COMMON /DONNEES/ A1,A2,A3
R = (A1+A2)/A3
RETURN
END
```

Instruction EQUIVALENCE

Cette instruction permet de faire partager à deux variables le même espace mémoire. Sa syntaxe est :

```
EQUIVALENCE (Var1,Var2), . . .
```

Exemple

```
CHARACTER A*4,B*4,C(2)*3
EQUIVALENCE (A,C(1)),(B,C(2))
```

Le schéma mémoire est représenté sur la figure 3.1.

Il faut toutefois faire deux remarques.

– Il n'est pas possible de forcer une variable à occuper deux emplacements mémoire.

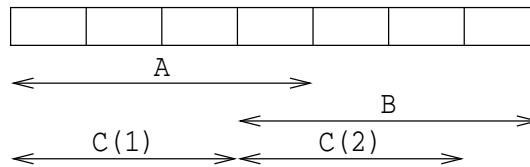


FIG. 3.1 – Schéma de mémoires partagées.

```

REAL R, S(2)
EQUIVALENCE (R, S(1)), (R, S(2))
– Il est impossible de spécifier que des éléments consécutifs ne soient pas stockés séquentiellement.
REAL R(3), S(2)
EQUIVALENCE (R(1), S(1)), (R(3), S(2))

```

Instruction *EXTERNAL*

Cette instruction permet de déclarer qu'une variable, un bloc fonctionnel ou une procédure ont été compilés dans un autre fichier que le fichier courant mais qu'ils seront disponibles lors de l'édition de liens (le *linkage*). Sa syntaxe est :

```
EXTERNAL Nom_externe
```

Instruction *PARAMETER*

Cette instruction permet de déclarer des constantes. Sa syntaxe est :

```
PARAMETER (Nom1=Const1, . . . )
```

Exemple :

```

REAL PI
PARAMETER (PI=3.1416)

```

3.2 Gestion des entrées/sorties

Cette partie est dédiée aux échanges de données entre programmes.

3.2.1 Fichiers de données

Un fichier de données est le moyen le plus simple pour échanger des données. Le format de ces dernières dépend des conventions adoptées par le programmeur.

Commande *OPEN*

Cette commande ouvre en lecture ou en écriture un fichier. Sa syntaxe est :

```
OPEN(U, FILE=Nom_fichier, STATUS=Status)
```

Où

- U est un entier représentant l'identificateur de fichier.
- Nom_fichier est une chaîne de caractères représentant le nom du fichier.
- Status est une chaîne de caractères pouvant prendre les valeurs suivantes :
 - OLD pour ouvrir un fichier qui existe déjà. Par exemple pour lire des données dans un fichier.
 - NEW pour ouvrir un nouveau fichier ou écraser un fichier dont les données ne sont plus utiles.
 - UNKNOWN. C'est la valeur par défaut. Si le fichier existe, il est ouvert, sinon, un nouveau fichier est créé.

Commande *REWIND*

Cette commande positionne la lecture ou l'écriture au début d'un fichier déjà ouvert. Sa syntaxe est :

```
REWIND (U)
```

Où U est l'identificateur du fichier.

Commande *CLOSE*

Cette commande ferme un fichier. Sa syntaxe est :

```
CLOSE (U)
```

Où U est l'identificateur du fichier.

3.2.2 Lecture et écriture dans un fichier

La lecture et l'écriture dans un fichier s'effectue grâce aux mêmes commandes que celles dédiées à l'affichage et à la saisie de données.

Commande *READ*

Cette commande lit des données dans un fichier et les stocke dans des variables. Sa syntaxe est :

```
READ (U, F, END=N) Liste_de_variables
```

Où

- U est l'identificateur du fichier.
- F est un numéro de référence définissant le format de lecture¹. Il est possible de ne pas définir de format en remplaçant F par un « * ».
- N est le numéro de référence de la ligne qui sera exécutée si le programme arrive à la fin du fichier.

Commande *WRITE*

Cette commande écrit des données dans un fichier. Sa syntaxe est :

```
WRITE (U, F) Liste_de_variables
```

Où

- U est l'identificateur du fichier.
- F est un numéro de référence définissant le format de lecture. Il est possible de ne pas définir de format en remplaçant F par un « * ».

Si le fichier existait déjà, les données sont écrasées.

3.2.3 Formatage des données

Il est possible de préciser le format de lecture ou d'écriture des données.

Déclaration *FORMAT*

Cette déclaration définit un formatage. Sa syntaxe est :

```
F FORMAT (Chaine1, Chaine2, . . . ChaineN)
```

Où

- F est le numéro de référence de ce formatage.
- Spec1, Spec2, . . . SpecN sont des chaînes de spécification de format.

¹Pour plus de détails, se reporter au chapitre 3.2.3

Règles de formatage

Les chaînes de spécification de format obéissent aux règles suivantes :

- ITaille décrit le formatage des entiers. Taille représente le nombre maximum de chiffres qu'il est possible de lire ou d'écrire (le signe doit être compris dedans).
- NombreX représente Nombre espaces.
- FTaille.Decimal décrit le formatage des rationnels. Taille représente le nombre maximum de chiffres qu'il est possible de lire ou d'écrire (en incluant le signe et/ou le point décimal). Decimal représente le nombre de chiffres après la virgule.
- ETaille.Exposant décrit le formatage des rationnels en notation scientifique. Taille représente le nombre maximum de chiffres qu'il est possible de lire ou d'écrire (en incluant le signe, le point décimal et la lettre « E »). Exposant représente le nombre de chiffres après le « E ».
- ATaille décrit le formatage des chaînes de caractères. Taille représente le nombre maximum de caractères qu'il est possible de lire ou d'écrire.
- LNombre décrit le formatage des booléens. Nombre représente le nombre de caractères des valeurs (si c'est 1, seul « T » et « F » sont utilisés).

Il est possible d'ajouter une chaîne de caractères entre des chaînes de spécification.

Pour répéter un formatage, il suffit de le faire précéder par un entier. Les parenthèses sont autorisées.

Exemple :

```
10   FORMAT(2(F10.2,' + i * ',F10.2)
```

pour la lecture de deux nombres complexes.

Caractères de contrôle

La première chaîne de caractères d'un format définit l'enchaînement entre les différentes lignes. Ainsi :

- ' ' passe à la ligne suivante.
- '0' passe à la ligne suivante et laisse une ligne vide.
- '1' revient en haut de l'écran.
- '+' continue sur la même ligne.

Attention ces caractères de contrôle ne sont pas pris en compte par tous les compilateurs.

3.3 Bibliothèque étendue de fonctions

Ce tableau récapitule un ensemble de fonctions qui sont « normalement » utilisables par n'importe lequel des compilateurs disponibles.

La première lettre d'une variable indique son type : C pour CHARACTER, G pour générique, I pour INTEGER, L pour LOGICAL, R pour REAL, et X pour COMPLEX.

| Commande | Description |
|------------------------|--|
| G = ABS(Gin) | Renvoie la valeur absolue ou le module de Gin. |
| R = ACOS(Rin) | Renvoie l'arc-cosinus en radian de Rin. |
| R = AIMAG(X) | Renvoie la partie imaginaire de X. |
| R = AINT(Rin) | Renvoie Rin tronqué. |
| R = ALOG(Rin) | Renvoie le logarithme népérien de Rin. |
| R = ALOG10(Rin) | Renvoie le logarithme décimal de Rin. |
| R = AMAX0(I1, I2, ...) | Renvoie l'argument de plus grande valeur. |
| R = AMAX1(R1, R2, ...) | Renvoie l'argument de plus grande valeur. |
| R = AMIN0(I1, I2, ...) | Renvoie l'argument de plus petite valeur. |
| R = AMIN1(R1, R2, ...) | Renvoie l'argument de plus petite valeur. |
| R = AMOD(R1, R2) | Renvoie le reste de la division de R1 par R2. |
| R = ANINT(Rin) | Renvoie Rin arrondi au plus proche entier. |
| R = ASIN(Rin) | Renvoie l'arc-sinus en radian de Rin. |
| R = ATAN(Rin) | Renvoie l'arc-tangente en radian de Rin. |

| Commande | Description |
|-----------------------------------|---|
| R = ATAN2 (R1, R2) | Renvoie l'arc-tangente en radian du nombre complexe (R1, R2). |
| R = CABS (X) | Renvoie le module de X. |
| X = CCOS (Xin) | Renvoie le cosinus de Xin. |
| X = CEXP (Xin) | Renvoie l'exponentielle de Xin. |
| C = CHAR (I) | Renvoie le caractère correspondant au code ASCII I. |
| X = CLOG (Xin) | Renvoie le logarithme népérien de Xin. |
| X = CMLX (G1, G2) | Crée le nombre complexe (G1, G2). |
| X = CONJG (Xin) | Renvoie le conjugué de Xin. |
| G = COS (Gin) | Renvoie le cosinus de Gin. |
| R = COSH (Rin) | Renvoie le cosinus-hyperbolique de Rin. |
| X = CSIN (Xin) | Renvoie le sinus de Xin. |
| X = CSQRT (Xin) | Renvoie la racine carrée de Xin. |
| R*8 = DABS (Rin*8) | Renvoie le module de Rin. |
| R*8 = DACOS (Rin*8) | Renvoie l'arc-cosinus en radian de Rin. |
| R*8 = DASIN (Rin*8) | Renvoie l'arc-sinus en radian de Rin. |
| R*8 = DATAN (Rin*8) | Renvoie l'arc-tangente en radian de Rin. |
| R*8 = DATAN2 (R1*8, R2*8) | Renvoie l'arc-tangente en radian du nombre complexe (R1, R2). |
| R*8 = DBLE (G) | Renvoie un nombre de type REAL*8. |
| R*8 = DCOS (Rin*8) | Renvoie le cosinus de Rin. |
| R*8 = DCOSH (Rin*8) | Renvoie le cosinus-hyperbolique de Rin. |
| R*8 = DDIM (R1*8, R2*8) | |
| R*8 = DEXP (Rin*8) | Renvoie l'exponentielle de Rin. |
| G = DIM (G1, G2) | |
| R*8 = DINT (Rin*8) | Renvoie Rin tronqué. |
| R*8 = DLOG (Rin*8) | Renvoie le logarithme népérien de Rin. |
| R*8 = DLOG10 (Rin*8) | Renvoie le logarithme décimal de Rin. |
| R*8 = DMAX1 (R1*8, R2*8, ...) | Renvoie l'argument de plus grande valeur. |
| R*8 = DMIN1 (R1*8, R2*8, ...) | Renvoie l'argument de plus petite valeur. |
| R*8 = DMOD (R1*8, R2*8) | Renvoie le reste de la division de R1 par R2. |
| R*8 = DNINT (Rin*8) | Renvoie Rin arrondi au plus proche entier. |
| R*8 = DPROD (R1*8, R2*8) | Renvoie le produit de R1 par R2. |
| R*8 = DSIGN (R1*8, R2*8) | Renvoie le signe de R1*R2. |
| R*8 = DSIN (Rin*8) | Renvoie le sinus de Rin. |
| R*8 = DSINH (Rin*8) | Renvoie le sinus-hyperbolique de Rin. |
| R*8 = DSQRT (Rin*8) | Renvoie la racine carrée de Rin. |
| R*8 = DTAN (Rin*8) | Renvoie la tangente de Rin. |
| R*8 = DTANH (Rin*8) | Renvoie la tangente-hyperbolique de Rin. |
| G = EXP (Gin) | Renvoie l'exponentielle de Xin. |
| R*4 = FLOAT (I) | Renvoie le réel correspondant à I. |
| I*2 = IABS (Iin*2) | Renvoie le module de Iin. |
| I = ICHAR (C) | Renvoie le code ASCII correspondant au caractère C. |
| I*2 = IDIM (I1*2, I2*2) | |
| I*2 = IDINT (Rin*8) | Renvoie Rin tronqué. |
| I*2 = IDNINT (Rin*8) | Renvoie Rin arrondi au plus proche entier. |
| I*2 = IFIX (G) | Renvoie G tronqué. |
| I*2 = INDEX (Chaine, CSouschaine) | Recherche CSouschaine dans Chaine. |
| I*2 = INT (G) | Renvoie G tronqué. |
| I*2 = IIGN (I1*2, I2*2) | Renvoie le signe de I1*I2. |
| I*2 = LEN (Chaine) | Renvoie la longueur de Chaine. |
| L = LGE (Chaine1, Chaine2) | Renvoie .True. si Chaine1 est supérieure ou égale à Chaine2. |

| Commande | Description |
|-----------------------------|---|
| L = LGT(Chaine1, Chaine2) | Renvoie .True. si Chaine1 est strictement supérieure à Chaine2. |
| L = LLE(Chaine1, Chaine2) | Renvoie .True. si Chaine1 est inférieure ou égale à Chaine2. |
| L = LLT(Chaine1, Chaine2) | Renvoie .True. si Chaine1 est strictement inférieure à Chaine2. |
| G = LOG(Gin) | Renvoie le logarithme népérien de Gin. |
| G = LOG10(Gin) | Renvoie le logarithme décimal de Gin. |
| G = MAX(G1, G2, ...) | Renvoie l'argument de plus grande valeur. |
| I*2 = MAX0(I1*2, I2*2, ...) | Renvoie l'argument de plus grande valeur. |
| I*2 = MAX1(R1*4, R2*4, ...) | Renvoie l'argument de plus grande valeur. |
| G = MIN(G1, G2, ...) | Renvoie l'argument de plus petite valeur. |
| I*2 = MIN0(I1*2, I2*2, ...) | Renvoie l'argument de plus petite valeur. |
| I*2 = MIN1(R1*4, R2*4, ...) | Renvoie l'argument de plus petite valeur. |
| G = MOD(G1, G2) | Renvoie le reste de la division de G1 par G2. |
| I*2 = NINT(Rin) | Renvoie Rin arrondi au plus proche entier. |
| R = REAL(G) | Renvoie G sous forme de réel. |
| G = SIGN(G1, G2) | Renvoie le signe de G1*G2 |
| G = SIN(Gin) | Renvoie le sinus de Gin. |
| R = SINH(Rin) | Renvoie le sinus-hyperbolique de Rin. |
| R*4 = SNGL(Rin*8) | Renvoie Rin sous forme de réel simple précision. |
| G = SQRT(Gin) | Renvoie la racine carrée de Gin. |
| R = TAN(Rin) | Renvoie la tangente de Rin. |
| R = TAN(Rin) | Renvoie la tangente-hyperbolique de Rin. |

3.4 La fonction *RANDOM*

Cette fonction n'existe pas sur tous les compilateurs. Il est, des fois, possible d'utiliser des fonctions Unix disponibles parmi les bibliothèques standards du C.

Ainsi, après avoir initialisé le générateur aléatoire grâce aux procédures `time` et `srand`, il est possible d'utiliser la fonction `rand` qui renvoie un nombre aléatoire.

Exemple :

```

PROGRAM suite_aleatoire
C
C   Le cote obscur du Fortran
C
  INTEGER i
  INTEGER j
C
C   Generation de la graine
C
  CALL time(j)
  CALL srand(REAL(MOD(j,1000)))
C
C   Generation d'une suite aleatoire
C
  DO 10 i=1,10
    PRINT*, rand()
10  CONTINUE
  END

```

Pour ceux qui utilisent *f2c* (sur un Unix ou sur MSDos), il faut créer une petite librairie qui permettra de corriger des problèmes de naming entre *f2c* et *gcc*.

Copiez ces quelques lignes dans un fichier nommé : `rand.c`

```
/* Le cote obscur du Fortran */  
  
#include <stdlib.h>  
  
typedef long int integer;  
typedef float real;  
  
real rand_() {return(((float)rand())/RAND_MAX);}  
void srand_(integer *i) {srand(*i);}  
integer time_(integer *t) {return(time(t));}
```

Puis tapez

```
gcc rand.c -c
```

Un fichier `rand.o` devrait avoir été généré.

Pour compiler vos programmes, vous devrez procéder en deux étapes :

```
f2c veidor.f; gcc veidor.c -c  
gcc veidor.o rand.o -lf2c -lm -o veidor
```

Ou

```
g77 veidor.f -c  
g77 veidor.o rand.o -o veidor.exe
```


Chapitre 4

Un exemple de programme complet

On se propose de calculer l'inverse d'une matrice carrée (en supposant qu'il existe) de dimension $(n \times n)$. Pour cela, nous avons utilisé un algorithme relativement simple : le pivot de Gauss généralisé.

4.1 Solution d'un système linéaire

Cet algorithme permet de calculer la solution d'un système linéaire. C'est-à-dire, de déterminer le vecteur colonne x tel que :

$$Ax = b \quad (4.1)$$

Avec A matrice carrée inversible et b un vecteur colonne.

On a alors :

$$x = A^{-1}b$$

Ainsi, en calculant chacun des x lorsque b décrit la base canonique des vecteurs colonnes de taille n , il est possible de déterminer la matrice A^{-1} .

4.2 Pivot de Gauss

Revenons à la résolution d'un système linéaire.

Soit le système $S_k(1)$ de taille k :

$$S_k(1) \iff \begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,k}x_k = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,k}x_k = b_2 \\ \vdots \\ a_{k,1}x_1 + a_{k,2}x_2 + \dots + a_{k,k}x_k = b_k \end{cases}$$

La première partie de l'algorithme consiste à déterminer la ligne de pivot. Si le coefficient $a_{1,1}$ n'est pas nul, le pivot est tout de suite trouvé. Sinon, il faut rechercher un coefficient $a_{p,1}$ non nul. Un tel coefficient existe sinon la matrice A ne serait pas inversible. On peut alors, par permutation de la ligne 1 et de la ligne p , obtenir un système $S_k(2)$ tel que $a'_{1,1}$ soit non nul.

La seconde étape de l'algorithme consiste à réduire le système $S_k(1)$ au système $S_k(2)$ de taille $k-1$ par des combinaisons linéaires. Ainsi, il est possible d'annuler le premier coefficient de la ligne i en soustrayant la première ligne multipliée par le coefficient $a_{1,i}$ à la ligne i multipliée par le coefficient $a'_{1,1}$. En effectuant cette opération sur toutes les lignes du système on obtient :

$$S_k(2) \iff \begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,k}x_k = b_1 \\ 0 + a''_{1,1}x_2 + \dots + a''_{1,k}x_k = b'_2 \\ \vdots \\ 0 + a''_{k-1,1}x_2 + \dots + a''_{k-1,k}x_k = b'_k \end{cases}$$

C'est en fait l'algorithme :


```

C      Nom :
C      MULTIP
C
C      Fonctionallite :
C      Procédure de multiplication de matrices / vecteurs
C
C      Entree(s) :
C      A : matrice / vecteur
C      m, n : dimensions
C      B : matrice / vecteur
C      o, p : dimensions
C
C      Sortie(s) :
C      C : matrice / vecteur ( C = A*B )
C      -> : .T. produit effectue
C      .F. dimensions incompatibles entre A et B
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
LOGICAL FUNCTION MULTIP (A, m, n, B, o, p, C)
REAL A(10,10), B(10,10), C(10,10)
INTEGER m, n, o, p, i, j, k
REAL t
IF (n .NE. o) THEN
  MULTIP = .FALSE.
  RETURN
ENDIF
DO 320 i=1, m
  DO 310 j=1, p
    t = 0
    DO 300 k=1, n
      t = t + A(i, k) * B(k, j)
300    CONTINUE
    C(i, j) = t
310  CONTINUE
320 CONTINUE
MULTIP = .TRUE.
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Nom :
C      LIGNE
C
C      Fonctionallite :
C      Procédure de recherche de la ligne de pivot a
C      l'etape n
C
C      Entree(s) :
C      A : matrice carree
C      m : dimension de A
C      n : colonne de recherche
C
C      Sortie(s) :
C      -> : ligne de pivot (0 si celle-ci n'existe pas)
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
INTEGER FUNCTION LIGNE (A, m, n)
REAL A(10, 10)
INTEGER m, n, p
p = n
400 IF (.NOT. ((p.LE.m) .AND. (A(p, n) .EQ. 0)))
+ GO TO 410
  p = p + 1
  GO TO 400
410 CONTINUE
IF (p .GT. m) p = 0
LIGNE = p
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Nom :
C      DUPLIQ
C
C      Fonctionallite :
C      Procédure de duplication matricielle / vectorielle
C
C      Entree(s) :
C      A : matrice / vecteur
C      m, n : dimension
C
C      Sortie(s) :
C      B : matrice / vecteur
C
C
C      Nom :
C      ECHANG
C
C      Fonctionallite :
C      Procédure de permutation de deux ligne d'une
C      matrice ou d'un vecteur
C
C      Entree(s) :
C      A : matrice / vecteur
C      m, n : dimension
C      k, l : lignes a permuter
C
C      Sortie(s) :
C      B : matrice / vecteur
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE ECHANG (A, m, n, k, l, B)
REAL A(10,10), B(10,10)
REAL t
INTEGER m, n, k, l, j
CALL DUPLIQ(A, m, n, B)
DO 600 j=1, n
  t = A(k, j)
  B(k, j) = A(l, j)
  B(l, j) = t
600 CONTINUE
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C      Nom :
C      PIVOTE
C
C      Fonctionallite :
C      Procédure de permettant d'effectuer le pivot a
C      l'etape i
C
C      Entree(s) :
C      p : etape du pivot
C      A : matrice carree
C      m : dimension de A
C      B : matrice / vecteur
C      o : nombre de colonnes de B
C
C      Sortie(s) :
C      A : matrice
C      B : matrice / vecteur
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE PIVOTE (p, A, m, B, o)
REAL A(10, 10), B(10, 10)
INTEGER i, j, m, o, p
REAL piv, t
piv = A(p, p)
DO 720 i=p+1, m
  t = A(i, p)
  A(i, p) = 0
C
C      Boucle de pivot de A
C
  DO 700 j=p+1,m
    A(i, j) = A(i, j) - A(p, j)*t/piv
700 CONTINUE
C
C      Boucle de pivot de A
C
  DO 710 j=1,o
    B(i, j) = B(i, j) - B(p, j)*t/piv
710 CONTINUE

```

```

720 CONTINUE
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Nom :
C   TRIANG
C
C   Fonctionallite :
C   Procédure de résolution d'un système triangulaire
C
C   Entree(s) :
C   A : matrice triangulaire
C   m : dimension de A
C   B : matrice / vecteur
C   n : nombre de colonnes de B
C
C   Sortie(s) :
C   X : matrice / vecteur
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
SUBROUTINE TRIANG (A, m, B, n, X)
REAL A(10, 10), B(10, 10), X(10, 10)
INTEGER m, n, i, j, k
REAL t
DO 820 k=1, n
  X(m, k) = B(m, k) / A(m, m)
  DO 810 i=m-1, 1, -1
    t = 0
    DO 800 j=i+1, m
      t = t + X(j, k) * A(i, j)
800    CONTINUE
    X(i, k) = (B(i, k) - t) / A(i, i)
810    CONTINUE
820    CONTINUE
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Nom :
C   GAUSS
C
C   Fonctionallite :
C   Procédure de résolution d'un système linéaire
C
C   Entree(s) :
C   A : matrice carrée
C   m : dimension de A
C   B : vecteur
C
C   Sortie(s) :
C   X : vecteur
C   -> : .T. si le pivot s'est effectuée correctement
C   .F. si le système n'est pas soluble
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
LOGICAL FUNCTION GAUSS (A, m, B, X)
REAL A(10, 10), B(10, 10), X(10, 10)
INTEGER m, i, p
INTEGER LIGNE
i = 1
900 IF (i .GE. m) GO TO 910
  p = LIGNE (A, m, i)
  IF (p .EQ. 0) THEN
    GAUSS = .FALSE.
    RETURN
  ENDIF
  IF (i .NE. p) THEN
    CALL ECHANG (A, m, m, i, p, A)
    CALL ECHANG (B, m, 1, i, p, B)
  ENDIF
  CALL PIVOTE (p, A, m, B, 1)
  i = i + 1
  GO TO 900
910 CONTINUE
CALL TRIANG (A, m, B, 1, X)
GAUSS = .TRUE.
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Nom :
C   INV
C
C   Fonctionallite :
C   Procédure d'inversion matricielle
C
C   Entree(s) :
C   A : matrice carrée
C   m : dimension de A
C
C   Sortie(s) :
C   X : matrice inverse
C   -> : .T. si A a été inversée
C   .F. si A n'est pas inversible
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
LOGICAL FUNCTION INV (A, m, X)
REAL A(10, 10), B(10, 10), X(10, 10)
INTEGER m, i, j, p
INTEGER LIGNE
C
C   Création de B = Id(n)
C
DO 1010 i=1, m
  DO 1000 j=1, m
    B(i, j) = 0
1000    CONTINUE
    B(i, i) = 1
1010 CONTINUE
C
C   Inversion de A
C
i = 1
1020 IF (i .GE. m) GO TO 1030
  p = LIGNE (A, m, i)
  IF (p .EQ. 0) THEN
    INV = .FALSE.
    RETURN
  ENDIF
  IF (i .NE. p) THEN
    CALL ECHANG (A, m, m, i, p, A)
    CALL ECHANG (B, m, m, i, p, B)
  ENDIF
  CALL PIVOTE (p, A, m, B, m)
  i = i + 1
  GO TO 1020
1030 CONTINUE
CALL TRIANG (A, m, B, m, X)
INV = .TRUE.
RETURN
END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C   Nom :
C   DETERM
C
C   Fonctionallite :
C   Procédure de calcul de la valeur absolue du
C   déterminant
C
C   Entree(s) :
C   A : matrice carrée
C   m : dimension de A
C
C   Sortie(s) :
C   -> : valeur absolue du déterminant
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
REAL FUNCTION DETERM (A, m)
REAL A(10, 10), B(10, 10)
INTEGER m, i, p
INTEGER LIGNE
REAL det
i = 1
1100 IF (i .GE. m) GO TO 1110
  p = LIGNE (A, m, i)
  IF (p .EQ. 0) THEN
    DETERM = 0
    RETURN
  ENDIF
  IF (i .NE. p) THEN
    CALL ECHANG (A, m, m, i, p, A)
  ENDIF
  CALL PIVOTE (p, A, m, B, 1)
  i = i + 1
  GO TO 1100
1110 CONTINUE

```

```
1110 CONTINUE
    det = A(1, 1)
    DO 1120 i=2, m
        det = det*A(i, i)
```

```
1120 CONTINUE
    DETERM = det
    RETURN
    END
```


Annexe A

Références et licence

La dernière version est disponible à l'adresse <http://www.softndesign.org/?page=manuels/fortran.php>. Des versions Postscript et PDF y sont aussi disponibles.

Ce document est distribué sous licence GPL. Vous êtes autorisé à le copier et/ou le redistribuer intégralement ou en partie, à la seule condition que cette mention y reste présente et conformément aux dispositions de la Licence de Documentation Libre GNU telle que publiée par la Free Software Foundation ; version 1.2 de la licence, ou encore (à votre choix) toute version ultérieure.