FORTRAN 77

SOMMAIRE

NOTATIONS UTILISEES

Dans l'ensemble de ce document nous utilisons pour la syntaxe des instructions la notation suivante :

- tout ce qui est en **caractères gras** est obligatoire

- le caractère ∆ désigne :
  - dans les instructions au moins un espace obligatoire
  - dans les données des exemples exactement un espace

- les expressions placées entre crochets [] désignent des paramètres ou des options facultatifs

Ce polycopié a été écrit par B. DHALLUIN de l'Ecole Supérieure d'Ingénieurs de Marseille, puis annoté et enrichi par F. Golay.

I UN PEU D'HISTOIRE ....

Le FORTRAN est né en 1954 dans les laboratoires de IBM. C'est le premier langage évolué de programmation. L'objectif était de remplacer les langages trop proches de la machine et de concevoir des programmes portables d'une machine à une autre. A partir de cette version rudimentaire du langage, une première normalisation est sortie en 1964 : FORTRAN IV. Plusieurs améliorations ont conduit, vingt trois ans après sa naissance, à la version normalisée actuelle du langage : FORTRAN 77. Basé sur les concepts de la programmation structurée, FORTRAN 77 est muni de l'essentiel des structures de programmation et satisfait le soucis de rentabilité des investissements informatiques des laboratoires et entreprises.

Le FORTRAN est le langage par excellence de la programmation numérique. Son nom provient de "*FORMULA TRANSLATION*". Opérationnel depuis près d'un quart de siècle, ne soyez pas étonnés qu'il soit très répandu dans les domaines de la recherche et de l'industrie.

Ce langage suit les évolutions des matériels informatiques de plus en plus puissants. Afin de rendre le langage plus performant, un consortium entre constructeurs de matériel et développeurs de logiciels a été créé afin de définir une nouvelle norme : FORTRAN 90. Respectant la norme FORTRAN 77, cette norme prend en compte des instructions supplémentaires spécifiques à la programmation objet.

II PROGRAMME FORTRAN

FORTRAN est un langage compilé. Un code exécutable peut être composé de un ou plusieurs fichiers ASCII.

II-1 Etapes de création d'un programme FORTRAN

Pour pouvoir écrire un code FORTRAN quatre étapes sont nécessaires :

a) Edition
Cette étape permet de générer un fichier source. Pour cela on utilise un des éditeurs disponibles sur le système de développement. Sous unix l'éditeur pleine page standard est *vi*, mais son utilisation est complexe et la bonne connaisance de ses commandes nécessite une utilisation courante. Fort heureusement, il existe plusieurs éditeurs nettement plus conviviaux sous environnement MOTIF (équivalent de WINDOWS sur PC). Ils sont spécifiques au matériel de développement et nous ne les citerons pas ici.

b) Compilation
Cette étape consiste à vérifier que le source est correctement écrit c'est à dire qu'il respecte les règles de syntaxe et de grammaire du langage. Le résultat de cette étape est la création d'un fichier binaire : fichier objet. Chaque fichier source d'un programme peut être compilé séparément.

c) Edition de lien
Cette étape consiste à associer tous les fichiers objet (appartenant soit au système, soit à l'utilisateur) formant l'ensemble du programme. Une édition de lien crée un fichier binaire : fichier exécutable. Même si le programme est composé d'un seul fichier objet, cette étape est obligatoire.

d) Exécution

Pour lancer un programme FORTRAN, il faut que le fichier exécutable existe, donc que les deux étapes précédentes se soient déroulées correctement (sans erreurs). La syntaxe d'exécution d'un programme diffère suivant le système.

Les fichiers créés à la fin chacune de des opérations portent une extension standard. Une extension est spécifiée par une ou plusieurs lettres séparées du nom du fichier lui même par un point. Suivant le système d'exploitation, ces extensions peuvent différer.

II-2 Règles de programmation

1) Majuscules, minuscules
FORTRAN ne différencie nullement les majuscules et les minuscules. Une instruction peut être écrite indifféremment en minuscules ou en majuscules. Dans un même source, une variable peut indifféremment s'écrire en minuscules ou en majuscules. Seule "l'orthographe" différencie deux variables.

2) Caractères blancs
Les blancs (ou espaces) sont des caractères qui ne jouent aucun rôle et qui peuvent être ajoutés (correctement) à volonté dans une ligne pour en rendre la lecture plus aisée.

3) Format d'une ligne
Une ligne FORTRAN est divisée en trois champs. Il est impératif de respecter ce format au risque de générer des erreurs de compilation. Ils se décomposent de la façon suivante:

Champ étiquette :	Chiffres entre les colonnes 1 et 5
Champ suite :	Un caractère quelconque en colonne 6 les instructions qui suivront seront interprétées par le compilateur comme étant la suite des instructions de la ligne précédente
Champ instructions :	Instructions entre les colonnes 7 et 72

4) Commentaires

Plus un programme est commenté, plus il est lisible par d'autre et donc portable, adaptable, facile à maintenir,.... Bien que tout commentaire soit subjectif, il est vivement conseillé de commenter abondamment ses propres programmes.

Pour introduire une ligne de commentaire (donc non compilée), il faut placer le caractère C à la colonne 1.

Il faut noter qu'une ligne complètement vide peut aider à la lisibilité du programme, mais le compilateur va jusqu'à la colonne 72 pour s'apercevoir que cette ligne est inutile. Pour accroître la rapidité de compilation autant placer le caractère C en première colonne de toute ligne vide. On ne perdra rien en lisibilité.

5) Exemple type
Nous donnons ici un exemple type de programme dans lequel toutes les règles précédentes sont appliquées. Il commence par deux lignes de commentaires permettant de connaître la numérotation des colonnes (padding) et donc de vérifier la concordance entre les champs et l'écriture du corps du programme.

```
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
C================================================================================
C                  Exemple de programme type : calcul d'un sinus
C================================================================================
C
       PROGRAM TYPE
C Declarations
       INTEGER I,      IDEUX
       REAL    AMPLITUDE(10),T
     &                 FREQUENCE, pi
C Initialisations
       FREQUENCE = 50.
       PI = 3.1416
       IDEUX = 0
C Corps du programme
       DO I=1,10
          ideux = IDEUX+2
          T = FLOAT(IDEUX)
          AMPLITUDE(I) = SIN (2. * PI*FREQUENCE*T)
          WRITE(*,100)IDEUX
      ENDDO
100   FORMAT(1X,I2)
      END
```

III CONSTITUTION D'UN PROGRAMME FORTRAN

Un programme FORTRAN est constitué d'un seul programme principal et d'un ensemble de fonctions et/ou sous-programmes répartis sur un ou plusieurs fichiers source. Chaque programme ou sous-programme doît être terminé par l'instruction END.

III-1 L'alphabet

Les dix chiffres décimaux
Les 26 lettres de l'alphabet
Les caractères spéciaux :

Espace ou blanc	
Egalité	=
Double cote	"
Simple cote	'
Symboles arithmétiques	+ - * / **
Paranthèses	()
Virgule	,
Point	.
Double point	:

III-2 La grammaire

1) Mots-clé
Ils constituent le vocabulaire du langage et servent à définir les déclarations de variables et les instructions : INTEGER, REAL, PARAMETER, DO, ENDDO, IF,.....

2) Opérandes
Ils constituent les noms symboliques que choisit le programmeur pour désigner les noms de variables et de procédures (sous-programmes ou fonctions). Ils doivent respecter trois règles :

- contenir uniquement des caractères alphanumériques
- ne jamais commencer par un caractère numérique
- être composés d'au maximum 6 caractères.

Cette dernière règle est définie comme telle dans la norme 77. Or, il s'avère que pratiquement tous les compilateurs ont largement augmenter ce maximum. Si l'on veille à écrire un programme portable, il faut impérativement respecter cette règle.

3) Séparateurs
Ils constituent la ponctuation du langage:

L'espace	
Virgule	,
Parenthèses	(____)
Slashes	/____/

III-3 Les opérateurs

Ce sont les actions qui s'exercent sur les opérandes. Ils sont de quatre types : unitaire, arithmétiques, relationnels, logiques. Plusieurs opérateurs de nature différente peuvent être utilisés dans une même expression. Il faut donc connaître la priorité de chacun. Celui de plus faible priorité est effectué en premier.
Cependant, si l'on ne retient pas les priorités de chaque opérateur, il est toujours possible de placer chaque opération élémentaire entre parenthèses. L'écriture gagne beaucoup en lisibilité.

1) Opérateur unitaire : l'affectation
C'est l'opérateur d'affectation d'une variable qui est le signe égal (=). Il s'applique de la droite vers la gauche. Dans l'exemple suivant on affecte la valeur 3 à la variable I: I = 3

2) Opérateurs arithmétiques

Exponentiation	**	(Priorité 1)
Multiplication	*	(Priorité 2)
Division	/	(Priorité 2)
Addition	+	(Priorité 3)
Soustraction	-	(Priorité 3)

3) Opérateurs relationnels

Ils sont tous de priorité 4. Ils sont une suite de deux caractères standard placés entre deux points sans espace :

.LT.	Lower Then ...	Strictement plus petit que ...
.GT.	Greater Then...	Strictement plus grand que...
.LE.	Lower or Equal...	Plus petit ou égal que...
.GE.	Greater or Equal...	Plus grand ou égal que...
.EQ.	EQual...	Egal à...
.NE.	Non Equal...	Non égal à...

4) Opérateurs logiques

.NOT.	Négation logique	(Priorité 5)
.AND.	Et logique	(Priorité 6)
.OR.	Ou logique	(Priorité 7)

IV DONNÉES

IV-1 Types de données

FORTRAN reconnaît cinq types de données :

1) Entier

Un entier est un signe (+ ou -) suivi d'une chaîne de digits décimaux
exemples : 324, +8901, -4

2) Réel

Un réel peut avoir deux formes d'écriture :

- la forme de base : un signe suivi d'une chaîne de digits décimaux contenant un point séparant la partie entière de la partie décimale .
exemples : 3.1415, -3. , +345.12698, -.008

- la forme exponentielle : c'est la forme de base suivie de la lettre E puis d'un signe et d'une suite de décimaux exprimant l'exposant décimal.
exemples : 31415.E-03, -0.3E01, +3.4512698E2, -8.E-2

3) Complexe

Tout complexe se représente sous la forme d'un couple de réels représentant respectivement les parties réelle et imaginaire. La syntaxe est entre parenthèses séparés par une virgule. Chacun peut être écrit suivant une des deux formes de réels.
exemples : (1., 2.E-5) , (0.,1.) , (2.7,678.E-4)

4) Chaîne de caractères

Une chaîne de caractères est délimitée par des simples côtes (apostrophes)
exemples : 'cours de FORTRAN' , ' Nom :', 'A+B = '

Il existe un opérateur particulier concernant les chaînes de caractères. Cet opérateur réalise la fonction de concaténation de deux chaînes. La syntaxe de cet opérateur est : // (double slash)
exemple de concaténation : STRING = 'AB'//'DEF'
Le résultat obtenu sera STRING = 'ABCDEF'

5) Logique
Seulement deux valeurs sont possibles. Positionnées entre deux points sans espace, ces valeurs sont :

.TRUE. vrai
.FALSE. faux

IV-2 Déclarations de données

IV-2-1 - Déclarations explicites

En FORTRAN, le type d'une donnée est défini explicitement par une instruction de déclaration. La déclaration peut se faire en simple (par défaut) ou double précision pour les entiers, les réels et les complexes. La précision donne la taille mémoire (en octets) que prend la variable. Le tableau ci-dessous donnent la liste des instructions de déclaration du FORTRAN pour chaque type de données.

Type de données	Instructions de déclaration	Simple précision	Double précision
Entier	INTEGER	INTEGER*2	INTEGER*4
Réel	REAL	REAL*4	REAL*8
Complexe	COMPLEX	COMPLEX*8	COMPLEX*16
Chaîne de caractères	CHARACTER*len len est le nbre de caractères de la chaîne		
Logique	LOGICAL		

exemple : INTEGER CPT, NBR
REAL*4 X, Y, A
REAL*8 TEMPS

IV-2-2 - Déclarations implicites

Une des particularités du FORTRAN, est qu'il existe des déclarations de données implicites suivant le nom des variables. Ainsi, si aucune spécification contraire n'est rencontrée , les déclarations implicites sont :

- toute variable commençant par I, J, K, L, M , N est de type INTEGER
- toute variable commençant par une autre lettre est de type REAL

Pour éviter les problèmes de déclaration, nous conseillons de nommer les entiers en commençant par les lettres I,J,K,L,M ou N.

Cependant, il est possible de modifier cette règle; pour ce faire, on utilise l'instruction IMPLICIT placée obligatoirement en tête de programme ou sous-programme.

Syntaxe : **IMPLICIT** ∆ type ∆ **(**c1[-c2],**)**
c1 et *c2* : caractères alphabétiques
c1-c2 signifie l'ensemble des caractères alphabétiques compris entre c1 et c2

exemple : IMPLICIT INTEGER (I-L)
IMPLICIT REAL (a-h,m-y)
IMPLICIT COMPLEX (z)
REAL ZORRO
Ces trois lignes signifient que les variables dont le nom commence par les lettres I, J, K ou L sont de type entier; celles commençant par Z, sauf la variable *ZORRO* qui est un réel, sont de type complexe; les autres sont de type réel.

Comme l'exemple ci-dessus le montre, l'instruction IMPLICIT n'empêche pas de déclarer en clair une variable.

Certaines versions de FORTRAN permettent l'usage de l'instruction IMPLICIT NONE qui annule toutes les définitions par défaut. Toutes les variables doivent, dans ce cas, être déclarées explicitement. Cette dernière instruction n'étant pas dans la norme 77, il faut se méfier de son usage si l'on désire un programme portable.

IV-3 Tableaux et matrices

Une variable tableau ou matrice est associée à un espace contigüe de la mémoire qui contient les valeurs des éléments du tableau. La ou les dimensions d'un tableau doivent figurer lors de la déclaration de la variable tableau sous la forme d'une constante non signée entre parenthèses. Un tableau possède au maximum 7 dimensions. L'accés à un élément se fait par l'intermédiaire d'indices associés au nom de la variable tableau. **La première valeur du tableau est à l'indice 1** et la dernière est à l'indice correspondant à sa taille.
Il existe deux formes de déclaration de tableaux :

1) Instruction DIMENSION
L'instruction DIMENSION permet de définir la taille d'un tableau dont le type a été déclaré précédemment de façon explicite ou implicite.

Syntaxe : **DIMENSION** ∆ nomvariable **(**taille1[,taille2...,taille7]**)**

2) Définition de la dimension lors de la déclaration
Il est également possible de définir la taille d'un tableau ou d'une matrice, lorsque la déclaration de la variable est opérée. Ceci s'exprime par une constante entière non signée que l'on place entre parenthèses directement à la suite du nom de la variable concernée.

```
exemple :   REAL        X
            COMPLEX     Y
            INTEGER     ITAB(100),IMAT(100,5)
            DIMENSION   X(10),Y(100,10,2)
```

Dans cet exemple, *ITAB* est un tableau d'entiers de dimension maximale 100, *IMAT* est une matrice de dimensions 100x5, *X* est un tableau de réels de dimension max 10, et *Y* est une matrice de complexes de dimensions 100x10x2.

IV-4 Caractères et chaînes de caractères

Une chaîne composée d'au maximum *len* caractères se déclare de la façon suivante :

```
            CHARACTER*len     STRING
ou          CHARACTER STRING*len
```

Si *len* n'est pas spécifiée *STRING* est déclarée comme un seul caractère.

```
exemple :   CHARACTER LETTRE
            CHARACTER*20      STRING,STAB(10)
            CHARACTER A*2,B*10
```

Dans cet exemple, *LETTRE* est une variable caractère unique, *STRING*, *A* et *B* sont des chaînes de respectivement 20, 2 et 10 caractères, *STAB* est un tableau de 10 chaînes de 20 caractères chacune.

Notons que l'on peut accéder à n'importe quelle sous-chaîne d'une chaîne complète en précisant la position dans la chaîne des premier et dernier caractères de la sous-chaîne :
STRING(2:6) désigne la sous-chaîne de *STRING* du 2ième au 6ième caractère inclus.

IV-5 Constantes et initialisation de variables

1) Constantes
Une ou plusieurs constantes utilisées dans un programme ou un sous-programme doivent être définies avant ou aprés suivant l'usage les déclarations par l'instruction PARAMETER :

Syntaxe : **PARAMETER**∆ **(**nom_constante = valeur,.....**)**

exemple :
```
REAL*4      PI
REAL*8      DPI
PARAMETER  (PI=3.1415926536, DPI=3.141592653589793D0)
```

2) Initialisation de variables

Toute initialisation de variable doit être précédée de sa déclaration, soit implicitement soit explicitement. Elle peut se faire par une affectation (signe égal), dans le corps du programme mais peut également être effectuée dans la partie déclaration :

Syntaxe : **DATA** ∆ nom_variable / valeur_initiale/

exemple :
```
INTEGER     nbpoint
REAL  rayon, temps(3)
DATA  temps /1.E-3, 0.2, 1./
DATA  rayon,nbpoint/10.,100/
```

Dans cet exemple, la variable *rayon* est initialisée à 10. et le nombre de points *nbpoint* à 100. Le tableau *temps* de dimension 3 est initialisé comme suit :

temps(1) = 1.E-3
temps(2) = 0.2
temps(3) = 1.

IV-6 Variables locales, variables globales

En FORTRAN, toutes les variables déclarées dans une unité de programme sont des **variables locales** à cette unité. Les **variables globales** sont placées dans une zone mémoire accessible par chaque unité qui l'utilise en plaçant l'instruction COMMON aprés le bloc de déclaration.

Syntaxe : **COMMON** ∆ /nom_bloc/var1[,var2,...]

Le COMMON définit une zone mémoire utilisée parallèlement par plusieurs unités de programme. Pour cela il doit être répété dans chaque sous-programme ou fonction qui utilise les variables définies comme appartenant à cette zone mémoire. Il peut y avoir plusieurs COMMON dans un programme.

exemple :
```
C       Programme principal
        REAL    X(100)
        CHARACTER*20   CAR
        INTEGER MAT(100,100)
        COMMON/TOTO/X,MAT,CAR
        .
        .
               Appel à sous-programme
        .
        .
        END

C       Sous-programme
        REAL    ALPHA(100)
        CHARACTER*20   CAR
        INTEGER RES(100,100)
        COMMON/TOTO/ALPHA,RES,CAR
        .
        .
        .
        RETURN
        END
```

Dans cet exemple, le COMMON de nom *TOTO* est répété dans chaque unité de programme utilisant les variables qui le contiennent. Il faut remarquer que les noms de variables du COMMON *TOTO* ne sont pas les mêmes dans le programme principal et dans le sous-programme. Il est impératif cependant que l'ordre d'écriture de la liste des variables soit de déclaration et de dimension identiques. Ainsi le contenu de la variable *MAT* du programme principal est celui de la variable *RES* dans le sous-programme. De même le contenu de *X* correspond au contenu de *ALPHA*.

V STRUCTURES DE CONTRÔLE

V-1 Boucles de contrôle

V-1-1 - Instructions DO , CONTINUE , ENDDO

Le rôle d'une boucle DO est identique à celui de FOR en PASCAL ou en C . Elle sert à gérer des calculs répétitifs. La syntaxe peut être de deux formes :

DO Δ etq Δ icpt = idebut**,**ifin[,ipas] Instructions etq **CONTINUE**	**DO** Δ icpt = idebut,ifin[,ipas] Instructions **ENDDO**

ou

icpt : compteur de boucle
idebut : première valeur que prendra *icpt*
ifin : dernière valeur que prendra *icpt*
ipas : pas d'incrémentation entre *idebut* et *ifin*. Si *ipas* n'est pas spécifié la valeur de l'incrément par défaut est 1
etq : étiquette pointant sur une instruction qui ferme la boucle.

Les paramètres *ictp*, *idebut*, *ifin* et *ipas* doivent être OBLIGATOIREMENT déclarés comme INTEGER. Par habitude, les noms des variables de compteur de boucle *icpt* commencent par I, J ou K.
La valeur de *ipas* peut être négative. La boucle effectue alors une décrémentation du compteur de boucle et il faut que *idebut* soit supérieur à *ifin*.
Il ne faut jamais modifier le compteur de boucle *icpt* à l'intérieur de la boucle(bloc *Instructions*). Seule l'instruction DO doit gèrer ce compteur.
Si *idebut* est égal à *ifin*, le bloc *Instructions* est exécuté une seule fois.
Dans le cas où *ipas* est positif, si *idebut* est supérieur à *ifin*, le bloc *Instructions* n'est pas exécuté. Le cas *ipas* négatif s'exprime de façon duale.

```
exemple :       INTEGER ICPT,DEB,FIN
                REAL    PRODAB(10,10), A(10,10) ,B(10,10)
                REAL    Y(100),OMEGA,T
                PARAMETER (OMEGA=314.16)
        C
        C       Calcul d'un produit matriciel PRODAB= A*B
        C
                DO I=1,10
                        DO J =1,10
                                PRODAB(I,J) = 0.
                                DO K = 1,10
                                PRODAB(I,J) = PRODAB(I,J) + A(I,k)*B(K,J)
                                ENDDO
                        ENDDO
                ENDDO
        C
        C       Calcul d'un sinus
        C
                T = 0.
                DEB = 100
                FIN = 1
                DO 10 ICPT=DEB,FIN, -2
                        Y(ICPT) = SIN(OMEGA*T)
                        T = T+1.
         10     CONTINUE
                END
```

S'il n'y a pas un nombre entier de pas d'incrémentation entre *idebut* et *ifin*, la dernière valeur du compteur de boucle *icpt* dans la boucle est la dernière valeur inférieure à *ifin*. A la sortie de la boucle, la valeur de *icpt* est la première valeur immédiatement supérieure à *ifin*. Ce dernier cas est illustré par l'exemple suivant :

```
DO I=1,50,2
:
:
ENDDO
```

Dans la boucle, la première valeur de *I* est 1 et sa dernière valeur est 49.
Aprés la boucle, la valeur de *I* est de 51.

V-1-2 - Instructions DOWHILE , CONTINUE , ENDDO

Une boucle DOWHILE permet de réaliser un ensemble d'instructions tant qu'une certaine condition est vraie. Comme précédemment, il existe deux syntaxes possibles :

DO Δ etq Δ **WHILE** **(**condition**)**	ou	**DOWHILE** Δ **(**condition**)**
Instructions		Instructions
etq **CONTINUE**		**ENDDO**

etq : étiquette pointant sur une instruction qui ferme la boucle
condition : expression logique

Si *condition* est fausse avant de rentrer dans la boucle, le bloc *Instructions* ne sera pas exécuté.

```
exemple :               LOGICAL        LTEST
                        INTEGER        I1,I2,A(100),B(100)
                        REAL           T,TFIN,OMEGA,ALPHA(100)
                        PARAMETER      (TFIN=10.,OMEGA = 314.)
                C
                C       Calcul d'un sinus
                C
                        T = 0.
                        I1 = 1
                        DOWHILE(T.LT.TFIN)
                               ALPHA(I1) = SIN(OMEGA*T)
                               T = T+0.5
                               I1 = I1+1
                        ENDDO
                C
                C       .Permutation de deux tableaux
                C
                        I1 = 1
                        I2 = 100
                        LTEST = .TRUE.
                        DO 20 WHILE(LTEST)
                               A(I1) = B(I2)
                               I1 = I1 + 1
                               I2 = I2 - 1
                               IF(I1.GT.100.)LTEST=.FALSE.
                 20     CONTINUE
                        END
```

V-2 Contrôles conditionnels - instruction IF

Tout comme en PASCAL ou en C, les contrôles conditionnels sont réalisés par l'instruction IF. Il existe plusieurs syntaxes de contrôles conditionnels :

IF (condition**)** Instruction_simple
condition : expression logique

Dans ce cas, une seule instruction est effectuée si *condition* est vrai.

IF (condition**) THEN**
Instructions
ENDIF

Dans ce cas, si *condition* est vrai alors l'ensemble du bloc *Instructions* est effectué.

IF (condition**) THEN**
Instructions_0
ELSE
Instructions_1
ENDIF

Dans ce cas, si *condition* est vrai alors l'ensemble du bloc *Instructions_0* est effectué sinon c'est le bloc *Instruction_1* qui est exécuté.

IF (condition_0**) THEN**
Instructions_0
ELSEIF (condition_1**) THEN**
Instructions_1
ELSEIF (condition_2**) THEN**
Instructions_2
.
.
.
[**ELSE**
Instructions_N]
ENDIF

Dans ce cas, si *condition_0* est vrai alors seul le bloc *Instructions_0* est exécuté, sinon si *condition_1* est vrai alors seul le bloc *Instructions_1* est exécuté, et ainsi de suite.... Si aucune condition n'est vérifiée alors seul de bloc *Instructions_N* est exécuté. La dernière instruction ELSE est facultative dans cette syntaxe.

exemple :

```
      LOGICAL        LTEST
      REAL           X,Y,Z,PI
      PARAMETER      (PI=3.1416)
      .
      .
      X = -1.
      IF(LTEST) X= 0.
      .
      .
      IF(X.LT.0.)THEN
             Y = PI/2.
             X = +1.
      ENDIF
      .
      .
      IF(X.EQ.0.)THEN
             Y = PI
      ELSE
             Y = 0.
      ENDIF
      .
      .
      IF(X.LT.0.)THEN
             Y = PI/4.
      ELSEIF(X.GT.0.)THEN
             Y = PI/2.
      ELSE
             Y = PI
      ENDIF
```

Cet exemple sans signification montre toutes les syntaxes possibles de l'instruction IF.

VI UNITÉS DE PROGRAMME

VI-1 - Programme principal

Un programme FORTRAN commence toujours par le programme principal. Le nom (facultatif mais conseillé) du programme est donné aprés l'instruction PROGRAM. Il se termine toujours par l'instruction END.

1) Instruction PROGRAM

Syntaxe : **PROGRAM** Δ nomp

nomp : nom du programme en respectant les mêmes règles que pour les noms de variables

Cette instruction permet de nommer le programme principal.

2) Instruction END

Cette instruction permet d'informer le compilateur sur la fin d'une unité de programme. Ce qui suit une instruction END concerne, s'il y a lieu, une autre unité de programme qui est forcément un sous-programme ou une fonction.

3) Instruction STOP

L'instruction STOP indique l'arrêt d'exécution du programme. Elle peut être placée n'importe où dans le programme, suivant la logique du problème.

4) Structure générale d'un programme principal

PROGRAM nompp

DECLARATION DES VARIABLES (locales au programme principal)
DEFINITION DE CONSTANTES (locales au programme principal)
DEFINITION DE VARIABLES GLOBALES

Jeu d'instructions du Programme principal

END

VI-2 - Sous-programmes

Un sous-programme est une unité de programme commençant par l'instruction SUBROUTINE et se terminant toujours par l'instruction END indiquant au compilateur une fin d'unité. Aussi, une seule instruction END doit figurer dans un sous-programme.

1) Instruction SUBROUTINE

Syntaxe : **SUBROUTINE** ∆ nomsp[**(**arg1[**,**arg2,...]**)**]

nomsp : nom que l'on désire donner au sous-programme. Il doit respecter les mêmes règles que les noms de variable
arg_i : arguments ou paramètres passés dans le sous-programme qui sont des noms de variable simple ou tableau qui doivent respecter les règles précédemment définies

Les arguments peuvent jouer trois rôles :

- argument d'entrée : paramètre passé qui ne sont pas modifié dans le sous-programme
- argument de sortie : paramètre initialisé et calculé (ou modifié) à l'intérieur du sous-programme
- argument d'entrée/sortie : paramètre passé et modifié dans le sous-programme

ATTENTION : les déclarations et dimension des arguments à l'intérieur du sous-programme doivent être identiques dans l'unité d'appel.

2) Instruction RETURN
Cette instruction doit apparaître au moins une fois dans tout sous-programme ou fonction. Son rôle est de restituer le contrôle à la unité de programme appelante (programme principal, sous-programme ou fonction).

3) Appel de sous-programme : instruction CALL
L'appel à un sous-programme se fait à l'intérieur ne n'importe quelle unité de programme.

Syntaxe : **CALL** ∆ nomsp[**(**arg1[,arg2,...]**)**]

nomsp : nom du sous-programme appelé
arg_i : arguments ou paramètres passés dans le sous-programme qui sont des noms de variable (simple ou tableau), des constantes ou des expressions arithmétiques.

Un sous-programme peut évidemment en appeler un autre, mais ne peut jamais s'appeler lui-même. La notion de récursivité n'existe pas en FORTRAN 77. Lors de son exécution, l'instruction CALL a pour effet de donner le contrôle au sous-programme.

4) Structure générale

SUBROUTINE nomsp**(**arg**,** ...**)**

DECLARATION DES VARIABLES (locales au sous-programme)
DEFINITION DE CONSTANTES (locales au sous-programme)
DEFINITION DE VARIABLES GLOBALES

Jeu d'instructions du Sous-programme
avec éventuellement des instructions RETURN

RETURN
END

5) Exemple

L'exemple ci-dessous effectue le calcul de la trace d'une matrice *MAT* de dimensions 100x100 et donne le résultat dans *RES*.

```
              SUBROUTINE TRACE(RES,MAT)
              REAL    RES,MAT(100,100)
C
              RES = 0.
              DO I=1,100
                     DO J = 1,100
                            IF(I.EQ.J)RES = RES +MAT(I,J)
                     ENDDO
              ENDDO
              RETURN
              END
```

VI-3 - Fonctions

Une fonction est analogue à un sous-programme. On distingue deux types de fonctions. Celles qui correspondent à des formules mathématiques courantes telles que sinus, racine carée, log néperien,... Elles sont incluses dans le langage lui-même et prote des noms normalisé; c'est à dire que quelque soit le système d'exploitation sur lequel on se trouve, la même fonction mathématique a le même nom. Elles sont toutes placées dans une bibliothèque de programme. Ces fonctions sont dites "intrinsèques".

Le programmeur peut estimer que cette bibliothèque est insuffisante et qu'il aimerait avoir la même souplesse d'utilisation pour des fonctions plus personnelles. Ces fonctions sont dites "utilisateurs".

VI-3-1 - Fonctions "utilisateurs"

1) Instruction FUNCTION

Syntaxe : [déclaration] **FUNCTION** ∆ nomfnc**(**arg1[**,**arg2,...]**)**

nomfnc : nom que l'on désire donner à la fonction. Il doit respecter les mêmes règles que les noms de variable et doit être éventuellement précédé de sa déclaration explicite de type

arg_i : arguments ou paramètres passés dans le sous-programme qui sont des noms de variable tels que définis précédemment

Par rapport à l'instruction SUBROUTINE , elle présente deux différences :
- il doit y avoir au moins un argument de passage en entrée
- le nom de la fonction désignant également le résultat, il est nécessaire de lui attribuer un type. Cette déclaration se fait soit de façon implicite (en fonction de la première lettre du nom) soit de façon explicite comme indiqué dans la syntaxe ci-dessus.

2) Instruction RETURN
Sa syntaxe et son utilisation sont les mêmes que dans le cas des sous-programmes.

3) Appel à une fonction
L'appel d'une fonction se fait en désignant son nom dans n'importe quelle expression arithmétique.

Syntaxe : val = nomfnc(arg1[,arg2,...])

val est la variable qui prendra la valeur de la fonction avec les paramètres passés. Si la fonction *nomfnc* possède un type de déclaration autre que celui découlant des règles implicites, il est absolument nécessaire de mentionner celui-ci dans les déclarations de type de toute unité de programme appelante.

4) Structure générale

FUNCTION nomfc(arg, ...)

DECLARATION DES VARIABLES (locales à la fonction)
DEFINITION DE CONSTANTES (locales à la fonction)
DEFINITION DE VARIABLES GLOBALES

Jeu d'instructions de la fonction calculant la valeur de *nomfc* avec éventuellement des instructions RETURN

RETURN
END

5) Exemple
L'exemple ci-dessous effectue le calcul N! (factoriel de N)

```
      INTEGER FUNCTION FACTORIEL(N)
      INTEGER N
C
      FACTORIEL = 1
      DO I=1,N
            FACTORIEL  = FACTORIEL * I
      ENDDO
      RETURN
      END
```

VI-3-2 - Fonctions intrinsèques

Les noms des fonctions intrinsèques sont des noms symboliques et en général :
- le nom d'une fonction entière commence par un I
- le nom d'une fonction double précision commence par un D
- le nom d'une fonction complexe commence par un C
- le nom d'une fonction complexe double précision commence par CD

Leur utilisation est identique à celle des fonctions "utilisateur" décrite au paragraphe précédent. Nous donnons ci-dessous une liste non exhaustive des fonctions les plus usuelles. Pour chacune, nous spécifions son nom et son type et celui de son ou ses paramètres.

Nom mathématique	Nom d'appel	Nbre para	Type des paramètres	Type de la fonction
Racine carrée	SQRT	1	REAL	REAL
Sinus	SIN	1	REAL	REAL
Cosinus	COS	1	REAL	REAL
Log Népèrien	ALOG	1	REAL	REAL
Log décimal	ALOG10	1	REAL	REAL
Tangente	TAN	1	REAL	REAL
Valeur absolue	IABS	1	INTEGER	INTEGER
	ABS		REAL	REAL
Conversions de type	SNGL	1	REAL	INTEGER
	FLOAT	1	INTEGER	REAL
	DBLE	1	REAL	DOUBLE PRECISION

VII ENTRÉES / SORTIES

Transférer des informations d'un périphérique (écran, console, imprimante, fichier,...) vers la machine ou inversement est une action d'entrée/sortie.

VII-1 Introduction

1) Unité logique
En FORTRAN tout périphérique est repéré par un entier; on parle alors d'unité logique.

Les numéros d'unité logique clavier et écran dépendent du système sur lequel on développe. Généralement, l'unité logique représentant le clavier est 5, et celle de l'écran est 6 par défaut .
Mais, si l'on ne veut pas retenir ces numéros, on peut placer le caractère *étoile* (*) pour une opération d'entrée/sortie sur ces deux unités logiques. Ainsi, un ordre de lecture au clavier aura la syntaxe :

READ(*,etq)X

De même un ordre d'écriture à l'écran sera de la forme :

WRITE(*,etq)I,J,K

où *etq* représente une étiquette relative au format que nous verrons plus tard.
Nous utiliserons les numéros d'unité 5 pour le clavier et 6 pour l'écran dans la suite du document.

Pour tout autre type de périphérique, l'utilisateur a la possibilité de choisir lui-même le numéro d'unité logique qu'il souhaite lui attribuer.

2) Enregistrement, format
Les entrées/sorties ne sont jamais réalisées octet par octet, mais par blocs d'octets que l'on nomme enregistrement. Tout enregistrement se termine toujours par les deux caractères ASCII <CR> (Carriage Return : retour du curseur en début de ligne) et <LF> (Line Feed : passage à la ligne).
Le contenu d'un enregistrement est défini par un format d'écriture ou de lecture.

VII-2 - Syntaxe générale des instructions d'entrée/sortie

Pour réaliser une opération d'entrée/sortie, il faut spécifier :
- la liste des variables concernées par l'opération
- l'unité logique correspondant au périphérique choisi
- la localisation des informations dans l'enregistrement concerné et la façon de les décoder ce qui est le rôle de l'instruction non exécutable FORMAT .

La syntaxe générale est :

READ ou **WRITE**	(lu,etq)	listv

etq **FORMAT** (lists)

lu : numéro d'unité logique repérant le périphérique. Ce peut être une constante ou une variable
etq : étiquette permettant de faire le lien entre l'instruction de lecture ou d'écriture et le format associé
listv : liste des variables (simples, indicées ou tableaux) que l'on désire lire ou écrire. Le nom d'un tableau est équivalent à la liste de tous ses éléments
lists : liste des spécifications élémentaires associées aux éléments de *listv*. Nous verrons par la suite comment sont définies ces spécifications.

1) Instruction FORMAT
Toute instruction FORMAT n'est valable que dans une seule unité de programme. Elle doit obligatoirement être précédée par une étiquette. Elle peut être placée à n'importe quel endroit de l'unité de programme pourvu que ce soit aprés les déclarations et avant l'instruction END. Une même instruction FORMAT peut être utilisée par plusieurs ordres READ ou WRITE.
Toutes les spécifications à l'intérieur d'une instruction FORMAT doivent être séparées par des virgules.

2) Instruction READ
Dès l'exécution de l'ordre READ sur l'unité logique correspondant au clavier le programme attendra que l'utilisateur ait validé son choix par appui sur la touche {RETURN} qui génère un <CR> et <LF>.
Si l'ordre READ est effectué sur une unité logique de type fichier l'enregistrement sera lu bloc par bloc suivant le format spécifié, jusqu'à ce que les caractères <CR> et <LF> soient atteints.

```
exemple :          READ(5,100) A,B
            100    FORMAT(....,...)
```

ceci signifie : lire les variables *A* et *B* sur l'unité logique 5 (le clavier par défaut), suivant les spécifications fournies dans l'instruction d'étiquette 100 qui ne peut être une autre instruction qu'un FORMAT.

3) Instruction WRITE
Sur l'unité logique écran, l'ordre WRITE effectue l'écriture de l'enregistrement demandé et passe à la ligne. L'écran interprète les deux caractères spéciaux <CR> et <LF>.
Il en va de même pour une unité logique de type fichier.

```
exemple :          WRITE(IOUT,2000) X,N,Z
            2000   FORMAT(...,...,..)
```

ceci signifie : écrire dans l'unité logique *IOUT* (qui doît être correctement initialisée) les variables *X*, *N* et *Z* suivant les spécifications fournies dans le format d'étiquette 2000.

```
                   REAL  TAB(4)
                   :
                   :
                   WRITE(6,50)TAB
            50     FORMAT(.........)
```

ceci signifie : écrire sur l'unité logique 6 (écran) les quatre éléments du tableau *TAB*.

Ce dernier exemple peut également s'écrire :

```
                   WRITE(6,51)TAB(1),TAB(2),TAB(3),TAB(4)
            51     FORMAT(..,..,..,..)
```

Il est important de noter dores et déjà que les spécifications de format des étiquettes 50 et 51 ne seront pas les mêmes.

VII-3 Spécifications de format

Nous allons décrire, dans ce paragraphe, un certain nombre de spécifications permettant de constituer la liste *lists* apparaissant dans l'instruction FORMAT.
Pour décrire la syntaxe de ces spécifications nous utiliserons la notation suivante :

- *w* désignera le nombre total de caractères (caractères "point" et "signe" compris) utilisés pour représenter une variable sur l'unité logique.
- *d* désignera (pour des réels uniquement) le nombre de décimales aprés le point
- *n* représentera un facteur de répétition facultatif qui devra être une constante sans signe.

VII-3-1 - Spécifications numériques

Ce sont celles qui permettent de lire ou d'écrire des variables de type numérique : entier, réel simple ou double précision.

1) Spécification I
Elle est utilisée pour des variables de type entier.

Syntaxe : [n]Iw

```
exemple :   INTEGER      K1,IX,IT
            :
            READ(5,1000)K1,IX,IT
```

```
1000    FORMAT(I3,I2,I4)
```

Si l'utilisateur tape : ΔΔ5181032, les valeurs respectives 5, 18 et 1032 seront attribuées aux variables *K1*, *IX* et *IT*.
S'il rentre : ΔΔ5Δ181032, les valeurs de K1, IX et IT seront respectivement 5, 1 et 8103.

2) Spécification F
Elle est utilisée pour des variables de type réel simple précision.

Syntaxe : [n]Fw.d

Il faut différencier ici écriture et lecture :

a) pour une écriture le nombre est écrit sous la forme :

```
±xxxxxxxxxxx.xxxxx
                 d
               <---->
  w
<------------------------>
```

Le signe + est souvent omis.

exemple :
```
        REAL Z,X,Y
        :
        X = 14.27
        Y = 28.6
        Z = -3.124
        WRITE(6,120)Z
        WRITE(6,115)X,Y
115     FORMAT(F6.1,F4.0)
120     FORMAT(F6.3)
```

Le résultat conduira à l'affichage suivant :

```
-3.124
```

```
Δ Δ 14.3Δ 29.
```

Notons que la spécification d'écriture des variables X et Y n'est pas suffisante en nombre de caractères. Dans ce cas l'écriture est arrondi au dernier digit d'écriture supérieur.

b) pour une lecture:
- si le point est présent dans la donnée, alors la valeur de *d* n'est pas utilisée. Ainsi, les caractères 12.34 lus en F5.4 conduira bien à 12,34 qui est celle que l'on obtiendrait également en lisant en F5.2 ou en F5.0 .
- si le point n'est pas présent dans la donnée, alors la valeur de *d* est effectivement utilisée. Ainsi les caractères 12345 lus en F5.2 conduiront à la valeur 123,45; Lus en F5.4, la valeur obtenue sera 1,2345

exemple :
```
        REAL   A,B,C
        :
        READ(5,10)A
        READ(5,15)B,C
10      FORMAT(F6.3)
15      FORMAT(F5.1,F5.3)
```

Si l'utilisateur tape dans un premier temps 12.45 puis {RETURN} la variable *A* aura pour valeur 12,45. Si l'utilisateur rentre ensuite la ligne suivante :
235.8Δ Δ 35800{RETURN}
les variables *B* et *C* auront pour valeur respective : 235,8 et 35,8

Il est donc préférable de TOUJOURS placer un point dans une donnée de type réel.

3) Spécification E
Elle est utilisée pour des variables réelles simple précision écrites en notation exponentielle.

Syntaxe : [n]Ew.d

De même que précédemment, il faut différencier écriture et lecture :

a) pour une écriture le nombre est écrit sous la forme :

```
±0.xxxxxxxxxxxE±yy
                d
   <------------>
  w
<------------------------>
```

Le signe + de la partie décimale est souvent omis. On remarque que *w* doit forcément être supérieur à *d+6*. Le 6 provenant des caractères ± de la partie décimale, 0, le point, E, ± de la partie exposant et au moins un caractère numérique pour l'exposant. Si cette condition n'est pas vérifiée, le nombre écrit risque d'être codé incorrectement sur l'unité logique et donc illisible.

```
exemple :          REAL  X(2)
                   :
                   X(1) = -14.27
                   X(2) = 28.6
                   WRITE (6,1234)X(1),X(2)
              1234 FORMAT(E11.4,E8.1)
```

Le résultat d'affichage sera alors :

```
-0.1427E+02Δ 0.3E+02
```

b) pour une lecture le nombre est écrit sous la forme :

```
±xxx.xxxxxxxxxxxE±yy
                 d
   <-------------->
  w
<------------------------>
```

L'utilisateur possède une grande liberté pour rédiger les nombres. Les signes + (celui de la partie décimale et celui de l'exposant) peuvent être omis.

- si le symbole *E* est présent dans le nombre, il est bien sûr pris en compte. Si le point est présent dans la partie décimale, il est considéré, sinon, on suppose qu'il y a *d* décimales. Ainsi, 12.34E+1 lu en E8.2 ou E8.3 conduit à la valeur 123,4 ; le point ayant été pris en compte indépendamment de la valeur de d. Par contre 12345E+1 lu en E9.2 conduit à 123.45E1 soit 1234,5; lu en E9.3 on obtiendrait 12.345E1 soit 123,45.

- si le symbole *E* n'est pas présent dans le nombre, le format Ew.d devient équivalent à Fw.d. Ainsi 12345 lu en E6.2 conduit à 123,45 .

```
exemple :          REAL  X,T
                   :
                   READ (5,100)X,T
              100  FORMAT(E8.2,E7.3)
```

Si l'utilisateur tape exactement : 67.89E-12345E-4{RETURN}
les variables *X* et *T* auront les valeurs respectives : 6,789 et 2,345E-4

4) Spécification D

Cette spécification est utilisée pour des variables réelles double précision écrites en notation exponentielle.

Syntaxe : [n]Dw.d

Son rôle est identique à la spécification Ew.d, le symbole *E* est remplacé par *D*. On retrouve exactement les mêmes mécanismes que pour la spécification E. Il faut noter qu'il est possible de trouver un exposant matérialisé par un E dans une donnée lue par une spécification Dw.d, sans que ceci ne génère une erreur d'exécution.

VII-3-2 - Spécifications alphabétiques

Ce sont celles qui permettent de lire ou d'écrire des variables de type chaîne de caractères.

1) Spécification Hollerith

Syntaxe : wHtexte

Utilisée uniquement en écriture, elle permet d'insérer la suite des *w* caractères de *texte* dans l'unité logique d'écriture. Le texte suivant le symbole H doit contenir exactement *w* caractères. Ceci peut être un risque important d'erreurs.
Nous préférons la formulation suivante :

'texte'

écrite directement dans l'instruction FORMAT.

```
exemple :       REAL  X
                X = 12.34
                FORMAT(' Δ LA Δ VALEUR Δ DE Δ X Δ EST Δ : Δ ', F5.2)
```

Le résultat d'affichage sera : Δ LA Δ VALEUR Δ DE Δ X Δ EST Δ : Δ 12.34

2) Spécification A

Syntaxe : [n]Aw

Elle permet de lire ou d'écrire une chaîne de *w* caractères.

exemples :

```
    CHARACTER TITRE*10
    TITRE = ' Δ CALCUL Δ : Δ'
    WRITE (6,11)TITRE
11      FORMAT(A10)

Résultat d'affichage :
  Δ CALCUL Δ : Δ
```

```
        CHARACTER C,NOM*5
        :
        READ(5,120)C,NOM
120     FORMAT(A1,A5)

Si l'utilisateur tape la ligne :
M Δ TOTO{RETURN}
Les variables C et NOM auront pour
valeur respective :
        'M'     et      ' Δ TOTO'
```

VII-3-3 - Spécifications élémentaires

1) Spécification X

Syntaxe : nX

Elle est utilisée pour avancer de *n* caractères dans l'unité logique d'entrée ou sortie.

exemples :

```
        INTEGER     I
        REAL  R
        :
        I = 5
        R = 42.3
        WRITE(6,2500)I,R
2500    FORMAT(1X,I1,5X,F5.1)

le résultat d'affichage sera :
Δ 5Δ Δ Δ Δ ΔΔ 42.3
```

```
        INTEGER K,L,M
        :
        READ(5,1200)K,L,M
1200    FORMAT(I2,2X,I1,I2)

Si l'utilisateur tape la ligne :
12345678{RETURN}
Les variables K,L et M auront pour
valeur respective :
12      5       et      67
```

Il faut noter que lors d'un ordre d'écriture, le premier caractère est interprété comme un caractère de contrôle par l'unité logique. Il faut donc placer une spécification de type *X* en début de liste de format d'écriture.

2) Spécification T

Syntaxe : Tp

Elle est utilisée pour se placer directement à une position donnée *p*.

exemples :

```
        INTEGER     N,K
        :
        N = 5
        K = 122
        WRITE(6,2000)N,K
2000    FORMAT(I2,T5,I3)

le résultat d'affichage sera :
Δ 5Δ Δ 122
```

```
        INTEGER K,L
        :
        READ(5,1200)K,L
1200    FORMAT(I2,T6,I2)

Si l'utilisateur tape la ligne :
12345678{RETURN}
Les variables K et L auront pour
valeur respective :
        12      et      67
```

3) Spécification / (Slash)

Syntaxe : [n](/)

Cette spécification permet de séparer deux enregistrements. Elle génère un <CR><LF> en écriture et lit un <CR><LF> en lecture.

exemples :

```
        INTEGER      N,K
        :
        N = 5
        K = 122
        WRITE(6,2000)N,K
2000    FORMAT(I2,2/,I3)

le résultat d'affichage sera :
Δ 5
{RETURN}
{RETURN}
122
```

```
        INTEGER K,L
        :
        :
        READ(5,1200)K,L
1200    FORMAT(I2,/,/,I2)

L'utilisateur devra taper deux
{RETURN} successifs pour rentrer la
deuxième valeur.
```

3) Spécification $ (Dollar)

Syntaxe : $

Cette spécification permet d'annuler en écriture, la génération d'un <CR><LF> en fin d'enregistrement. Autrement dit, elle aura pour effet de maintenir le curseur en fin de ligne écrite. Le symbole $ se place généralement à la fin de la liste *lists* d'une instruction FORMAT.

VII-4 Répétition d'un groupe de spécifications

VII-4-1 - Répétition de spécifications simples

Nous avons vu que lorsque l'on avait besoin de plusieurs spécifications identiques, on pouvait éviter de les répéter, en utilisant un "facteur de répétition" pour cetaines d'entre elles.
Ainsi la spécification 3I5 est équivalente à I5,I5,I5

De même, il est posible de répéter un groupe de spécifications.
Ainsi la spécification 3(2X,I5) est équivalente à 2X,I5,I2X,5,2X,I5

VII-4-2 - Boucles implicites de spécifications

Nous allons mentionné ici, la façon de prendre en compte des tableaux à une ou plusieurs dimensions, avec une seule spécification.
Jusqu'ici nous avons mentionné qu'une liste *listv* de l'instruction READ ou WRITE était composée de variables simples ou indicées. Nous généralisons la notion de liste de la manière suivante :

lists = {variables simples, variables indicées, (tableaux(i),i=d,f[,p])}

- *i* : indice de parcours du tableaux
- *d* : valeur initiale de l'indice *i*
- *f* : valeur finale de l'indice *i*
- *p* : pas d'incrémentation entre deux valeurs d'indice (s'il est omis sa valeur est de 1)

Ainsi (A(K),K=1,3) est équivalent à A(1),A(2),A(3)
et (A(J),B(J),J=1,5,2) est équivalent à A(1),B(1),A(3),B(3),A(5),B(5)

exemples :

a)
```
        REAL R(200)
        READ(5,1000)R
1000    FORMAT(E16.8)
```

Il faudra taper un {RETURN} aprés chaque valeur introduite soit 200 au total.

b)
```
        REAL R(200)
        READ(5,1000)R
1001    FORMAT(10E16.8)
```

Dans ce cas, il faudra taper dix valeurs de *R* par ligne avant de confirmer par un {RETURN}. Vingt lignes seront nécessaires. On remarque que le format de lecture est réexploré automatiquement à chaque fois qu'il est épuisé.

c)
```
        REAL B(100)
        WRITE(6,1002)(B(I),I=1,100,4)
1002    FORMAT(5(1X,E16.8))
```

L'ordre WRITE indique qu'un élément de *B* sur quatre sera affiché. Compte tenu de la réexploitation automatique, l'ordre FORMAT indique qu'il y aura cinq valeurs séparées par un blanc par ligne

d)
```
        REAL A(15,5)
        DO I=1,15
                WRITE(6,1003)(A(I,J),J=1,5)
1003            FORMAT(5(1X,E16.8)
        ENDDO
```

Le résultat obtenu sera du type suivant :

```
A(1,1)  A(1,2)  A(1,3)  A(1,4)  A(1,5)
A(2,1). ......          .....           .....           ....
:               :               :               :               :
A(15,1) A(15,2) A(15,3) A(15,4) A(15,5)
```

Le même résultat est obtenu avec un même format, mais un ordre WRITE différent :

```
        WRITE(6,1003)((A(I,J),J=1,5),I=1,15)
```

e)
```
        REAL X(15,5)
        WRITE(6,1004)(I,(A(I,J),J=1,5),I=1,15)
1004    FORMAT(I3,1X,5(1X,E16.8)
```

Le résultat obtenu sera du type suivant :

```
1       A(1,1)  A(1,2)  A(1,3)  A(1,4)  A(1,5)
2       A(2,1). ......          .....           .....           ....
        :               :               :               :               :
15      A(15,1) A(15,2) A(15,3) A(15,4) A(15,5)
```

VIII FICHIERS

Un fichier est une unité logique au même titre que l'écran ou le clavier. Les opérations de lecture ou d'écriture sont donc identiques à celles décrites au paragraphe précédent.

VIII-1 Différents types de fichiers

En FORTRAN, tout fichier est créé lors de son ouverture. Dès sa création, il faut spécifier sa structure et son accés.

a) Structure

Un fichier créé par un programme FORTRAN peut avoir deux structures différentes :

- texte ou ASCII : tout enregistrement sera écrit sous-forme ASCII. L'avantage est que la totalité du fichier sera lisible par n'importe quel dispositif d'affichage du système de développement : éditeur, listing, ...
- binaire : tout enregistrement est codé en binaire. Il ne sera lisible par aucun outil d'affichage du système. L'avantage de cette structure est que le fichier sera forcément plus petit en taille que pour la précédente.

b) Accès

L'accès à un fichier correspond au mode lecture ou d'écriture des enregistrements qu'il contient ou contiendra .Les deux accès principaux d'un fichier sont :

- Séquentiel : pour accéder à un enregistrement *n*, il faut avoir accédé au *n-1* précédent. L'avantage est que la gestion d'accès des enregistrements est automatique.
- Direct : les enregistrements sont repérés par un indice. Ainsi on peut accéder directement à un enregistrement donné. L'avantage porte sur la rapidité d'accès à un enregistrement, mais il faut devoir gérer les indices d'enregistrements.

VIII-2 Instruction OPEN

Cette instruction permet d'associer une unité logique à un fichier. De plus elle permet de définir le mode d'accès au fichier (séquentiel, direct) ainsi que sa structure (taille et format des enregistrements, ...) et certaines informations spécifiques (existence, erreur d'ouverture, protection en écriture,....).

Syntaxe : **OPEN(**KEYWORD=v,....**)**

Les paramètres à passer dans un ordre OPEN sont définis par un ensemble de mots-clé (*KEYWORD*) dont on donne la valeur *v*. Le type de la valeur *v* dépend du mot-clé qui lui est associé et peut être un entier ou une chaîne de caractères.
Nous ne donnons ici qu'un nombre minimal. En effet, la liste de l'ensemble des mots-clé serait trop longue à donner ici. Elle se trouve en ANNEXE II.

KEYWORD	v	type de v	Remarques
UNIT	numéro d'unité logique	Entier	Inférieur à 99
FILE	nom du fichier à ouvrir	Chaîne de caractères	
STATUS	'NEW' 'OLD'	Constante chaîne de caractères	-Création -Modification/lecture d'un fichier
ACCES	'SEQUENTIEL' (défaut) 'DIRECT'	Constante chaîne de caractères	Mode d'accés - Séquentiel - Direct
FORM	'FORMATTED' (défaut) 'UNFORMATTED'	Constante chaîne de caractères	Formattage des enregistrement
RECL	Longueur des enregistrements	Entier	En octets
ERR	Label d'erreur d'ouverture	Constante entière	Etiquette de saut si erreur d'ouverture

VIII-3 Instruction CLOSE

Cette instruction permet de dissocier une unité logique d'un fichier. Autrement dit, elle ferme le fichier. De même que OPEN, elle possède une série de paramètres que nous ne listerons pas ici. Le seul mot-clé obligatoire à donner est UNIT suivi du numéro d'unité logique que l'on désire fermer.

Syntaxe : **CLOSE(**KEYWORD=v,....**)**

VIII-4 Lecture et écriture d'un fichier

Les opérations de lecture et d'écriture d'un fichier sont des opérations d'entrée/sortie. Ces opérations sont réalisées par les instruction READ et WRITE, et il convient de se référer au chapitre précédent pour en connaître l'usage. Il faut malgré tout préciser que l'unité logique qui est le premier paramètre de ces deux instructions doit correspondre à celle utilsée lors de l'ouverture du fichier.

VIII-5 Exemple

Imaginons que l'on veuille représenter graphiquement plusieurs périodes d'un sinus. Pour cela nous allons calculer la valeur du sinus à intervalles de temps

réguliers et nous allons placer ces valeurs dans un fichier. Le format d'écriture du fichier sera le suivant : première valeur nombre de points total du fichier, et chaque enregistrement contiendra un premier réel qui sera l'instant de calcul, et un second qui sera la valeur du sinusà cet instant. Un programme graphique pourra ainsi lire ce fichier suivant le format défini ci-dessus.

Le programme type correspondant à ce cahier des charges simple aura l'allure suivante :

```
C23456789012345678901234567890123456789012345678901234567890123456789012
C
C                    Calcul d'un sinus et mise en fichier du résultat. On
calcule 20 points par
C                   période et on calcule au maximum NPMAX points
C
              PROGRAM CALCUL_SINUS
C
C Declarations
              INTEGER          I,IDEUX,NPMAX
              REAL             FREQUENCE, AMPLITUDE,PI,PHI
              REAL             T,TFIN,PAS
              PARAMETER(PI = 3.1416, NPMAX=100))
              REAL             VALEUR(NPMAX),TEMPS(NPMAX)
C
C Initialisations
              WRITE(6,10)
  10        FORMAT(1X,'Frequence (en Hertz) ? ')
              READ(5,20)FREQUENCE
  20        FORMAT(E14.7)
              WRITE(6,30)
  30        FORMAT(1X,'Amplitude ? ')
              READ(5,20)AMPLITUDE
              WRITE(6,40)
  40        FORMAT(1X,'Dephasage (en Radians) ? ')
              READ(5,20)PHI
              WRITE(6,50)
  50        FORMAT(1X,'Temps final (en Secondes) ? ')
              READ(5,20)TFIN
C
              T = 0.
              I = 0
              PAS = 1./(FREQUENCE*20.)
C
C Calcul du sinus
              DOWHILE (T.LE.TFIN .AND. I.LE.NPMAX)
                        I = I+1
                        T = T +PAS
                        VALEUR(I) = AMPLITUDE*SIN (2. * PI*FREQUENCE*T +
PHI)
                        TEMPS (I) = T
                        WRITE(6,60)TEMPS(I),VALEUR(I)
              ENDDO
              NP = I
 60         FORMAT(1X,E14.7,5X,E14.7)
C
C Enregistrement du fichier
              CALL ECRIRE(NPMAX,NP,TEMPS,VALEUR)
              END


.../...
C
C Sous-programme assurant l'enregistrement des résultats
C dans le fichier RES.DAT
C
              SUBROUTINE ECRIRE(MAX,NP,T,VALEUR)
C
              INTEGER  NP,MAX
              REAL     T(MAX), VALEUR(MAX)
C
C Creation et ouverture du fichier RES.DAT
              OPEN(UNIT=50, FILE='RES.DAT', STATUS ='NEW')
C
C Ecriture du fichier
              WRITE(50,10)NP
10          FORMAT(1X,I3)
              DO I=1,NP
                        WRITE(50,20)T(I),VALEUR(I)
              ENDDO
20          FORMAT(1X,E14.7,5X,E14.7)
C
C Fermeture du fichier
              CLOSE(UNIT=50)
              RETURN
              END
```

Puisqu'aucune option particulière n'est mentionnée dans l'instruction d'ouverture du fichier, ce sont celles par défaut qui sont prises en compte. RES.DAT est donc un fichier ASCII formatté.
L'avantage d'effectuer l'enregistrement du fichier dans un sous-programme est que si l'on a besoin de lire ce fichier, il suffira d'écrire un sous-programme de lecture qui sera le pendant de celui d'écriture. Il s'écrira ainsi :

```
C
C Sous-programme assurant la lecture des résultats situés dans le fichier
RES.DAT
C
              SUBROUTINE LECTURE(MAX,NP,T,VALEUR)
```

```
C
            INTEGER  NP,MAX
            REAL     T(MAX), VALEUR(MAX)
C
C Ouverture du fichier RES.DAT
            OPEN(UNIT=50, FILE='RES.DAT', STATUS ='OLD')
C
C Lecture du fichier
            READ(50,10)NP
10         FORMAT(1X,I3)
            DO I=1,NP
                    READ(50,20)T(I),VALEUR(I)
            ENDDO
20         FORMAT(1X,E14.7,5X,E14.7)
C
C Fermeture du fichier
            CLOSE(UNIT=50)
            RETURN
            END
```