Le langage Fortran 77

Michel Billaud <billaud@labri.u-bordeaux.fr> Août 1999

Support de cours pour l'enseignement de Fortran 77 en première année en IUT. Ecrit sous WordStar vers 1985-1987. Converti en SGML (DTD Linuxdoc) en août 1999, avec quelques légères corrections.

Table des matières

1 Le langage Fortran 77 **4**

1.1 Objets de base du Fortran 77 4

1.1.1 Jeu de caractères 4

1.1.2 Èléments du langage 4

1.1.3 Types de données, constantes 5

1.1.4 Variables, scalaires, tableaux 5

1.1.5 Unités de programme 6

1.2 Présentation des programmes 7

1.3 Instructions 8

1.3.1 Les déclarations de variables 8

1.3.2 Initialisation statique des variables 8

1.4 Les déclarations d'unités de programmes 9

1.4.1 Programme principal. 9

1.4.2 Sous-programmes et fonctions 9

1.4.3 Block Data 11

1.4.4 Instructions de contrôle 12

1.4.5 Entrées-Sorties séquentielles 16

2 Annexe : les nombres **21**

2.1 Les entiers 21

2.1.1 Les types 21

2.1.2 Fonctions courantes 21

2.2 Les réels 22

2.3 Types 22

2.4 Fonctions 22

2.5 La trigonométrie, et cie. 23

2.6 Conversions explicites de type 23

3 Annexe : utilisation des chaînes de caractères 24

3.1 Déclaration 24

3.2 Constantes 24

3.3 Opérations sur les chaines de caractères 25

3.3.1 Concaténation, affectation, sélection 25

3.3.2 Quelques fonctions 25

3.4 Entrées-sorties 25

4 Annexe : passage de Paramètres 26

4.1 Passage de paramètres 26

4.2 Passage de fonctions ou sous-programmes en paramètres 27

4.3 Tableaux Formels 28

4.4 Adresses de retour alternatives 28

5 Annexe : COMMON 30

6 Annexe : entrées-Sorties directes formattées 32

Chapitre 1

Le langage Fortran 77

1.1 Objets de base du Fortran 77

1.1.1 Jeu de caractères

- Alphabet Majuscule : A B C ... Z
- Alphabet Minuscule : a b c ... z
- Chiffres : 0 1 2 ... 9
- Caractères spéciaux : + - * / = ( ) < > . ; $ _ ' etc.

1.1.2 Èléments du langage

Noms d'instructions, mots réservés

```
INTEGER REAL CHARACTER DOUBLE PRECISION COMPLEX LOGICAL DO GOTO
IF THEN ELSE ELSEIF STOP END COMMON SUBROUTINE CALL RETURN FUNCTION
READ WRITE FORMAT OPEN CLOSE REWIND BACKSPACE DATA ...
```

Noms symboliques

- Les noms symboliques (identificateurs) sont des suites d'au plus 30 caractères (lettres, chiffres ou _). Le premier caractère doit être une lettre. Majuscules et minuscules sont confondues.
- Servent à nommer les variables, les zones communes et les unités de programme (programme principal, fonctions et sous-programmes) et les zones communes.
- La longueur des noms symboliques est limitée à 6 dans le cas des unités de programmes et des zones communes.

Opérateurs

- signes mathématiques + - * / ** (Exponentiation)
- opérateurs de relation :

```
.LT. ( < )    .GT.  ( > )
.LE. (  )     .GE.  (  )
.EQ. ( = )    .NE.  ( # )
```

- opérateurs logiques :

```
.NOT. (non)   .AND. (et)    .OR.  (ou)
.EQV. (équivalence)  .NEQV. (ou exclusif)
```

- concaténation de chaînes de caractères :

```
//
```

Expressions

Identificateurs et constantes reliés par des opérateurs. Exemples :

```
A + B * ( C ** D )
(1.LE.K).AND.(K.LE.MAX)
PATH // '/' // NOM // '.' // EXT
```

1.1.3 Types de données, constantes

- Type Entier : 123 -6123
- Type Réel : 1.414 65.4E-6
- Type Logique (Booléens) : .TRUE. .FALSE.
- Type chaîne : 'Hello' 'C"est l"heure'

Autres types de données : Réels double-précision, Complexes (simple et double précision), Fractionnels, Virgules fixes, etc ...

1.1.4 Variables, scalaires, tableaux

Variables

Une *variable* est une grandeur dont la valeur peut changer au cours de l'exécution d'un programme. Une variable est désignée par un *identificateur*. Un *type* est associé à chaque variable

- soit par déclaration explicite (exemple INTEGER A,B,C);
- soit implicitement selon la règle dite « IJKLMN » :

Toute variable dont le nom commence par I, J, K, L, M ou N et qui n'a pas été déclarée explicitement est par défaut de type INTEGER. Les autres variables non déclarées sont de type REAL.

Scalaires

Scalaire : donnée unique désignée par un nom.

Tableaux

Tableau : collection d'objets du même type.

Une déclaration de tableau se fait en mentionnant - après son nom - la liste des intervalles de ses indices.

Exemple :

```
REAL TAB(5:12,3,0:20)
```

déclare un tableau à 3 dimensions dont les éléments sont de type REAL. Le premier indice peut varier entre 5 et 12, le second entre 1 et 3, et le troisième entre 0 et 20. On peut utiliser jusqu'à 7 dimensions.

Elément de tableau : donnée élémentaire désignée par l'identificateur du tableau suivi par une liste d'indices placés entre parenthèses et séparés par des virgules. Le nombre d'indices doit correspondre au nombre de dimensions déclarées. Exemple : `TAB(6,2,0)`.

Pendant l'exécution, si un indice déborde des bornes indiquées, le résultat est indéfini.

1.1.5 Unités de programme

Un programme est composé d'une ou plusieurs unités de programme :

- un programme principal (commençant par PROGRAM ...),
- éventuellement des sous-programmes et des fonctions (commençant respectivement par SUBROUTINE ... ou FUNCTION ...),
- le cas échéant un « BLOCK DATA » (ou plusieurs, mais c'est déconseillé).

La fin de toute unité de programme est indiquée par la présence d'une instruction END .

La portée des identificateurs de variables et des étiquettes est limitée à une unité de programme.

Les noms des sous-programmes, fonctions et zones communes (COMMON) sont accessibles à partir de toute unité.

1.2 Présentation des programmes

- Les instructions FORTRAN sont écrites sur des lignes de 80caractères.
- Seules les 72 premières positions sont utilisées. Les colonnes 73 à 80 sont ignorées. Attention à ne pas déborder.
- Les lignes qui contiennent un C en première colonne sont des commentaires ; le reste de la ligne n'est pas pris en compte par le compilateur.
- Les autres lignes sont des lignes « instructions »; qui se décomposent en 3 zones :
  - la zone étiquette (de 1 à 5),
  - la zone suite (colonne 6),
  - la zone instruction proprement dite (7 à 72).

Une instruction particulièrement longue peut s'écrire sur plusieurs lignes consécutives; dans ce cas seule la première ligne peut comporter une étiquette, et les « lignes-suite » sont identifiées par un caractère quelconque différent du blanc et du zéro (on met en général un S, ou un chiffre).

Un exemple de programme FORTRAN :

```
          1         2         3         4         5
12345678901234567890123456789012345678901234567890123456789012345678901234567890
-----S--------------------------------------------------------
          PROGRAM DEGRAD
C
C Imprime une table de conversion degrés -> radians
C
          INTEGER DEG
          REAL    RAD,COEFF
C
C En-tête
          WRITE (*,10)
10        FORMAT (' ',20('*') /
     S              ' * Degres * Radians *' /
     S              ' ',20('*') )
C Corps
          COEFF = (2.0 * 3.1416) / 360.0
          DO 100 DEG = 0,90
               RAD = DEG * COEFF
               WRITE (*,20) DEG,RAD
20             FORMATF (' *  ',F4.1,'  * ',F7.5,' *')
100       CONTINUE
C
C Fin du tableau
C         WRITE(*,30)
30        FORMAT (' ',20('*') )
C
          STOP
          END
```

1.3 Instructions

1.3.1 Les déclarations de variables

Une variable est déclarée en citant : son type, son nom et, (pour les tableaux) les bornes des indices.

Principaux types :

- `INTEGER INTEGER*2 INTEGER*4`
- `REAL COMPLEX DOUBLE PRECISION`
- `LOGICAL`
- `CHARACTER*1 .... CHARACTER*256`

Exemple :

```
INTEGER*4      A(20,3),B
CHARACTER*10   C(0..9)
LOGICAL        D(3,4,8:12)
```

Ces instructions déclarent :

- Un tableau A de 20 x 3 entiers,
- Une variable scalaire entière nommée B,
- Un tableau C de 10 chaînes de 15 caractères,
- Un tableau D de 3 x 4 x 5 scalaires booléens.

1.3.2 Initialisation statique des variables

Il est possible de donner une valeur initiale aux variables d'une unité de programme pendant le chargement du programme. C'est ce qu'on appelle l'*initialisation statique* d'une variable, par opposition à l'*initialisation dynamique* qui se fait à l'exécution au moyen d'une instruction d'affectation.

Exemples :

```
REAL           PI,E
DATA           PI,E / 3.14159 , 2.718 /

CHARACTER*1    SPACE,BELL,ESCAPE
DATA           SPACE,BELL,ESCAPE / ' ',Z'07',Z'1B' /

CHARACTER*10   NAMES(10)
DATA           NAMES / 3*'AAAAAAAAAA',7*'XXXXXXXXXX' /
```

Le dernier exemple utilise des facteurs de répétition : les 3 premiers élements de NAMES contiendront 'AAAAAAAAAA', les 8 autres 'XXXXXXXXXX'.

1.4 Les déclarations d'unités de programmes

Rappelons que les noms d'unités de programmes sont limités à 6 caractères, et que le texte d'une unité de programme se termine sur une instruction END.

1.4.1 Programme principal.

Cette unité de programme commence par la ligne :

```
      PROGRAM nom
```

Exemple 1 :

```
      PROGRAM ESSAI
      INTEGER  I
C
      DO  1000 I=1,10
           WRITE  (*,*) ' Bonjour !'
1000  CONTINUE
C
      STOP
      END
```

1.4.2 Sous-programmes et fonctions

Exemple 2 : définition d'un sous-programme

```
      SUBROUTINE  ECHANG ( X,Y )
      INTEGER X,Y,INTER
C
      INTER = X
      X = Y
      Y = INTER
C
      RETURN
      END
```

Les sous-programmes sont appelés par l'ordre CALL. Exemple :

```
       CALL ECHANG( T(I) , T(I+1) )
```

Exemple 3 : Définition de fonction

```
      INTEGER FUNCTION  MAX (N1,N2)
      INTEGER  N1, N2
C
      IF (N1 .GE. N2) THEN
           MAX = N1
```

```
          ELSE
               MAX = N2
          ENDIF
     C
          RETURN
          END
```

Les fonctions sont appelées à partir des expressions. Elles retournent toujours une valeur.

Exemple d'appel de fonction :

```
I12 = 2 * MAX(A+1,B+2) - MAX (C,10)
```

- Les sous-programmes et fonctions doivent toujours contenir au moins une instruction RETURN ; l'exécution de cette instruction provoque le retour à l'unité de programme appelante. Il est possible d'avoir plusieurs instructions RETURN dans une même unité de programme (voir exemple 4).
- Les instructions de déclaration SUBROUTINE et FUNCTION servent à indiquer :
  - le nom de l'unité de programme,
  - le nombre de paramètres formels (Dans l'exemple 2, X et Y sont les paramètres formels de ECHANG),
  - pour les fonctions le type du résultat.

Exemple 4 : passage d'un tableau formel

```
          INTEGER FUNCTION INDEX (T,N,V)
          INTEGER  N, T(N), V, K
C
C Cette fonction calcule l'index de la première occurence de V
C dans le tableau T. (Retourne 0 si cette valeur est absente)
C
          DO  100 K = 1,N
               IF ( T(K) E.EQ. FV ) THEN
                    INDEX = K
                    RETURN
               ENDIF F
100       CONTINUE
C
          INDEX = 0
          RETURN
          END
```

IMPORTANT : lors de l'appel d'une unité de programme, les paramètres effectifs doivent être conformes - en nombre et en type - aux paramètres formels. (Chose que les compilateurs FORTRAN ne vérifient en général pas, et qui provoque des *gags* plutôt difficiles à déceler).

Exemple typique d'erreur :

```
REAL FUNCTION F (X)
REAL F, X
     F = X + 1.0
RETURN
END
```

Si l'on exécute alors l'instruction Y=F(0) (au lieu de F(0.0)), on récupère dans Y quelque chose qui est tout-à-fait différent de la valeur attendue 1.0. Essayez!

Autre source d'erreurs : ne pas oublier de déclarer le type d'une fonction appelée dans l'unité appelante. Exemple d'erreur :

```
INTEGER FUNCTION CARRE (N)
INTEGER N
     CARRE = N * N
RETURN
END

INTEGER FUNCTION P4 (N)
INTEGER  N
     P4 = CARRE ( CARRE ( N ) )
RETURN
END
```

Dans P4 la fonction CARRE est considérée comme rendant un résultat de type REAL (d'après la règle IJKLMN), ce qui n'est pas cohérent avec la définition de la fonction CARRE. Il faut donc ajouter dans P4 (et dans toute unité de programme qui appelle CARRE) la déclaration « INTEGER CARRE ».

1.4.3 Block Data

Cette unité de programme un peu spéciale sert uniquement à l'initialisation statique (par DATA) des zones communes. Elle commence par la ligne :

```
     BLOCKDATA nom
```

et ne peut contenir que :

- des déclarations de type (INTEGER, REAL ...)
- des déclaration de zones communes (COMMON),
- des initialisations statiques (DATA).

Une zone commune ne peut être citée que dans un seul BLOCKDATA. Il est d'ailleurs conseillé de n'avoir qu'un seul BLOCKDATA par programme.

(Pour plus d'informations se reporter à l'annexe sur l'utilisation des zones communes).

1.4.4 Instructions de contrôle

Affectation

Exemple : PAS (N , P) = PAS (N-1 , P-1) + PAS (N , P)

Le déroulement d'une affectation « var = expression » est le suivant :

- l'expression à la droite du signe '=' est évaluée selon son type propre,
- le résultat est au besoin converti dans le type de la variable de gauche,
- la valeur obtenue est rangée dans la variable.

Ainsi, si les variables I et R sont respectivement de type INTEGER et REAL, les instructions :

```
I = 1
R = I / 2.0
```

provoquent l'affectation de la valeur 0.5 à R, tandis que :

```
I = 1
R = I / 2
```

met 0.0 dans R (l'expression I/2 étant de type entier, la division est une division entière de résultat nul, converti ensuite en valeur réelle).

CONTINUE

C'est une «non-opération», qui sert uniquement à *poser une étiquette* (en particulier dans les boucles DO).

Exemple :

```
      DO 1000 K=1,N
           A(K) = B(K) + 1
1000  CONTINUE
```

STOP

Provoque l'arrêt complet du programme en cours.

GOTO

Exemple : GOTO 1234

Provoque un saut (branchement) à l'instruction (de la même unité de programme) qui porte l'étiquette 1234.

GOTO calculé

Exemple : GOTO (1010,2000,3000,1515),I+1

Si l'expression I+1 a pour valeur 1, alors saut à l'étiquette 1010. Si la valeur est 2, saut à 2000; etc.

Si la valeur calculée estt inférieure à 1 ou supérieure au nombre d'étiquettes citées, on passe simplement à l'instruction suivante.

IF logique

Instruction de la forme : IF (condition) instruction

Exemple :

```
IF  ( M .LT. FT(K) ) M = T(K)
K = K + 1
```

Si la condition citée (M plus petit que T(K)) est vraie, l'instruction située sur la même ligne est exécutée, puis on passe à la suivante (K = K + 1). Dans le cas contraire, l'instruction suivante est exécutée directement (on ne fait pas l'affectation M = T(K)).

Le « IFlogique » est très souvent employé en compagnie de l'instruction GOTO, c'est-à-dire sous la forme :

```
IF( condition ) GOTO etiquette
```

C'est le « GOTO conditionnel ».

Bloc IF

Forme générale :

```
IF ( condition1 ) THEN
     bloc 1
ELSE IF ( condition2 )  THEN
     bloc 2
ELSE IF ( condition3 )  THEN
     bloc 3
....
ELSE
     bloc-else
ENDIF
```

Si la condition 1 est vraie, le bloc 1 (suite d'instructions) est exécuté et on passe à l'instruction qui suit le ENDIF. Sinon, si la condition 2 est vraie on exécute le bloc 2, etc ...

Si les conditions sont toutes fausses, c'est le bloc-else qui est exécuté.

Le nombre de parties ELSE IF n'est pas limité (on peut ne pas en mettre). La partie ELSE est également facultative.

Exemples :

```
          PROGRAM CHOIX
          CHARACTER*1 CODE
C
C on demande le caractere CODE
C
1         WRITE (*,*) ' Tapez A,B,C,? ou X'
          READ (*,100) CODE
100       FORMAT F (A1)
C
C au besoin, on transforme les minuscules en majuscules.
C
          IF(  ('a'.LE.CODE).AND.(CODE.LE.'z')  THEN
               CODE  = CHAR ( ICHAR ( CODE ) - 32 )
          ENDIF
C
C selon CODE, on appelle FA,FC,FC,MENU et on recommence.
C si CODE=X on arrête tout.
C
          IF (CODE.EQ.'?') THEN
              CALL MENU
          ELSE IF ( CODE.EQ.'A' ) THEN
              CALL FA
          ELSE IF ( CODE.EQ.'B' ) THEN
              CALL FB
          ELSE IF ( CODE.EQ.'C' ) THEN
              CALL FC
          ELSE IF ( CODE.EQ.'X' ) THEN
               WRITE  (*,*) ' C''est tout !'
               STOP
          ELSE
               WRITE (*,*) ' Quoi ?'
          END IF
C
          GOTO 1
          END
```

« IF arithmétique »

Tout manuel FORTRAN qui se respecte se doit de citer ce vénérable vestige de la préhistoire informatique, période FORTRAN I ! Cette chose est appelée vulgairement « IF à trois pattes », d'après sa forme générale qui est :

```
IF ( Expression ) Etiquette1,Etiquette2,Etiquette3
```

- Si l'expression arithmétique entre parenthèses est négative (strictement), on saute à l'étiquette 1.
- Si elle est nulle on passe à l'étiquette 2.
- Enfin, si elle est strictement positive on saute à l'étiquette 3.

Cette forme est utile (et encore ...) uniquement si on a vraiment trois traitements distincts à effectuer dans chaque cas. L'exemple classique est la recherche des solutions d'une équation du second degré ... qui ne marche pas puisque les opérations réelles fournissent rarement un résultat exactement nul, à cause de la précision limitée. En réalité il faut faire une comparaison «à epsilon près».

Quoi s'il en soit, on fait exactement la même chose avec un GOTO conditionnel mille fois plus clair.

Cette chose a malheureusement survécu à toutes les révisions du langage FORTRAN, parce que beaucoup de programmes FORTRAN l'utilisaient, et que personne n'a le courage de les refaire. Notamment on rencontre des choses du genre :

```
IF (M-T(K)) 100,200,200
100   M = T(K)
200   K = K + 1
```

Le devoir sacré de tout programmeur FORTRAN est de remplacer illico tous les « IF arithmétiques » qu'il rencontre par des « IF logiques ». Ne pas corriger une faute, c'est là la véritable faute (Confucius).

DO

Exemple : DO 100 K=1,N

L'instruction DO sert à indiquer qu'un certain « bloc » d'instructions (le corps de la boucle) est à répéter pour plusieurs valeurs de l'indice de boucle (ici K), cet indice variant entre une borne inférieure (ici 1) et une borne supérieure (ici N).

Le corps de la boucle est composé des instructions quisuivent l'instruction DO jusqu'à (et y compris) l'instruction dont l'étiquette est indiquée (ici 100). Une bonne habitude consiste à utiliser une instruction CONTINUE pour poser cette étiquette, et à indenter (décaler) le corps de boucle.

Exemple

```
            SUBROUTINE  PROMAT (A,B,C,N,M,P)
C
            INTEGER   N,M,P
            REAL      A(N,M),B(M,P),C(N,P)
C
            INTEGER   I,J,K
            REAL S
C
            DO 300  I=1,N
```

```
                DO 200 J=1,P
                     S = 0.0
                     DO 300 K=1,M
                          S = S + A(I,K)*B(K,J)
100                  CONTINUE
                     C(I,J)=S
200             CONTINUE
300        CONTINUE
C
           RETURN
           END
```

Quelques remarques :

- A l'intérieur d'une boucle, il est fortement déconseillé de chercher à modifier la valeur de l'indice de boucle.
- Il est interdit de se brancher (par GOTO) de l'extérieur d'une boucle DO vers l'intérieur. Par contre, on peut très bien utiliser un GOTO pour s'échapper d'une boucle.
- Deux boucles DO sont soit emboitées, soit disjointes. En aucun cas elles ne peuvent se chevaucher.
- On peut préciser le pas, en utilisant la forme :

  ```
  DO étiquette  indice = début , fin , pas
  ```

  et aussi utiliser des boucles à indice de type réel (attention aux erreurs d'arrondis).

1.4.5 Entrées-Sorties séquentielles

Le point important est qu'un ordre d'entrée-sortie FORTRAN (lecture ou écriture) mentionne (au moins) trois choses :

- Le « numéro d'unité logique » du fichier sur lequel l'opération doit être effectuée;
- Le « format » décrivant la structure de l'enregistrementà lire ou écrire;
- La liste des variables concernées.

Il est également possible de préciser ce qu'il faut faire si l'ordre d'entrée-sortie se déroule mal, ou si on détecte une fin de fichier

WRITE

Exemples :

```
WRITE  ( 6,100 ) A,B,C
WRITE  ( 6,123 ) A, B+1 , ( T(K), K=1,N )
```

Le premier exemple signifie « Ecrire sur l'unité logique numéro 6 les contenus des variables A,B et C en utilisant la description d'enregistrement donnée par l'instruction FORMAT d'étiquette100 (située dans la même unité de programme) »

Le second exemple montre une « boucle DO implicite », qui parcourt les N premiers élements du tableau T.

Autre exemple de boucle DO implicite :

```
WRITE (6,421) (  ((10*I+J), J=1,I) , I=1,9 )
```

qui imprime la suite de nombre 11,21,22,31,32,33,41...99 sur l'unité logique et dans le format que vous devinez.

Le numéro d'unité logique 6 désigne -en général- la « sortie standard » (votre écran), on peut mettre une étoile * si on ne s'en souvient pas.

Si on ne désire pas spécifier de manière précise le format d'écriture, on met aussi une étoile à la place de l'étiquette de format.

Dans ce cas le compilateur se débrouille pour fabriquer un format convenable (de son point de vue!). C'est bien pratique lorsqu'on fait de la mise au point par « espionnage », du style :

```
SUBROUTINE XYZ (A,B)
WRITE (*,*) ' Je rentre dans XYZ avec A=',A,'et B=',B
     ...
WRITE (*,*) ' Je sors de XYZ avec A=',A,' et B=',B
RETURN
END
```

et aussi pour les programmes « quick'n dirty ».

READ

Exemples :

```
READ (5,100) X,Y,Z
READ (5,100,END=123) A,B,C
READ (5,100,END=123,ERR=987) U,V,W
```

Il n'est guère difficile de deviner la signification du premier exemple : « lire sur l'unité logique numéro 5 les nouvelles valeurs des variables X,Y et Z, selon le format 100 ».

Dans le second exemple l'option « END=123 » signifie qu'en cas de fin de fichier il faut se dérouter à l'instruction portant l'étiquette 123.

Le troisième exemple montre l'option « ERR=987 » qui indique : « si une erreur se produit pendant la lecture (par exemple une lettre dans une zone censément numérique) alors aller en 987 ».

Le numéro 5 est affecté à l'entrée standard (votre clavier) ; on peut mettre une étoile à la place. On peut aussi mettre une étoile au lieu du numéro de format, la lecture s'effectue alors en « format libre » ; c'est-à-dire que l'utilisateur tapera ses données séparées par des blancs ou des virgules, les chaines de caractères étant entourées par des quotes « ' ».

L'usage des formats libres est :

- une très bonne chose pour l'entrée des données en mode conversationnel;
- un gadget tout-à-fait douteux (voire nuisible) pour les lectures sur fichiers.

FORMAT

Exemple :

```
          WRITE (*,100) I,J,K
          ...
100       FORMAT ( 5X ,'Le ppcm de ',I2,' et ',I2,' vaut ',I4)
```

Dans cet exemple l'ordre WRITE provoque l'écriture (sur la sortie standard) d'une ligne composée de :

- 5 blancs (format 5X)
- la chaine 'Le ppcm de '
- la valeur de l'entier I, sur deux caractères (format I2)
- la chaine ' et '
- la valeur de l'entier J, sur deux positions (format I2)
- la chaine ' vaut '
- la valeur de l'entier K, sur 4 caractères (format I4).

L'instruction FORMAT (en fait c'est une déclaration) n'a de sens qu'utilisée par un ordre READ ou WRITE. Il y a donc nécessairement un numéro d'étiquette devant.

Quelques spécifications de format

- I : pour les entiers. Exemples I2 , I1 , I12 , etc..
- A : pour les chaines de caractères : la chaine 'ABCDEF'écrite avec le format A3 donne ABC (troncation à 3 caractères), avec le format A10 on obtient ABCDEF (cadrage à gauche sur 10positions).
- F : pour les réels en « virgule fixe ». Par exemple 3.1416 écrit avec le format F6.2 donne 3.14 (6 positions, dont 2 après le point décimal).

- E : pour les réels « avec exposant » : Par exemple 3.1416 en format E11.5 s'écrit .31416E-01 (En tout 11 positions, dont 5 pour la partie décimale, l'exposant Enn étant toujours sur les 4 caractères les plus à droite).
- X : pour laisser un blanc (ou ignorer un caractère en lecture).
- / : pour abandonner la ligne en cours et passer à l'enregistrement suivant.

Répétition de FORMAT

- La spécification « 5I4 » équivaut à « I4,I4,I4,I4,I4 » . (La même chose s'applique aux spécifications E,F,A,X ...).
- On peut faire répéter un groupe de spécifications en plaçant ce groupe entre parenthèses et en le faisant précéder par le facteur de répétition.

Exemple : « 5(I3,X) » équivaut à « I3,X,I3,X,I3,X,I3,X,I3,X ».

Il y a une règle très amusante qui s'applique au cas où l'on veut faire écrire plus de choses que le format n'en spécifie. Exemple :

```
          WRITE  (*,100) A,B,C
100       FORMAT  (2I4)
```

Cette règle dit :

> Si la parenthèse finale est rencontrée alors qu'il reste des données à traiter :
>
> - s'il n'y a pas de groupes de spécifications entre parenthèses, le format est repris au début;
> - s'il y a des groupes entre parenthèses, l'exploration reprend au début du groupe dont la parenthèse de droite a été rencontrée la dernière avant la parenthèse finale.

En réfléchissant un peu, c'est finalement très simple.

OPEN

```
OPEN  (1)
OPEN  (1,FILE='FICH.ABC')
OPEN  (1,FILE='TOTO.X',STATUS='OLD',ERR=123)
```

Le premier exemple est une ouverture simple du fichier numéro 1. Dans le second, on précise le nom du fichier que l'on veut associer au numéro d'unité logique 1 (ici FICH.ABC). Dans le troisième, on précise que le fichier que l'on veut ouvrir (TOTO.X) est censé exister déjà (OLD), et qu'en cas d'erreur (si fichier absent, donc) il faut aller à l'étiquette 123.

On peut donner comme STATUS la chaîne 'NEW', qui signifie que le fichier n'existait pas préalablement, ou 'UNKNOWN' ...

Il existe des quantités d'autres clauses, voir en particulier l'annexe relative à l'utilisation des fichiers en accès direct.

Il n'est pas nécessaire d'ouvrir les fichiers 5 et 6.

CLOSE

```
CLOSE (1)
```

ferme le fichier - préalablement ouvert - indiqué.

Il est inutile de fermer les fichiers standards 5 et 6.

Chapitre 2

Annexe : les nombres

2.1 Les entiers

2.1.1 Les types

type	occupation mémoire	intervalle
INTEGER	32 bits (16 utiles)	-32768..+32767
INTEGER*2	16 bits	-32768..+32767
INTEGER*4	32 bits	-2**31..+2**31

Les constantes entières ont le type INTEGER si elles sont dans l'intervalle comprises entre -32768 et +32767, sinon INTEGER*4.

2.1.2 Fonctions courantes

Les fonctions arithmétiques les plus courantes :

- MIN0 (.....) : minimum de plusieurs expressions
- MAX0 (.....) : maximum
- IABS (expr) : valeur absolue
- MOD (e1,e2) : reste de la division entière de e1 par e2.

Dans certains cas -assez particuliers, il est vrai- on peut avoir besoin de considérer les entiers comme des «chaînes de bits», et de faire des masquages, des décalages, des complémentations, etc ...

- NOT (expr) : Complémentation.
- IAND (e1,e2) : Produit (ET).
- IOR (e1,e2) : OU inclusif.

- IEOR (e1,e2) : OU exclusif.
- ISHFT(e1,e2) : Les bits de e1 sont décalés de e2 positions vers la gauche. Les positions vacantes sont comblées par des 0. Décalage à droite si e2 est négatif.
- IROT (e1,e2) : Rotation des bits de e1, de e2 positions vers la gauche.

Il peut être commode d'utiliser à cette occasion les constantes hexadécimales, qui sont de la forme Z'012345FEDC'. Exemple : IAND(I,Z'000F'), qui permet de récupérer (par masquage) les 4 bits de poids faible de la variable I.

2.2 Les réels

2.3 Types

type	occupation mémoire	mantisse
REAL	32 bits	24 bits
DOUBLEPRECISION	64 bits	56 bits

Ces deux types permettent de représenter des nombres compris (approximativement) entre 5.4*10**(-79) et 7.2*10**75 .

Exemple de constantes :

- REAL : 1. -3.14 6.28E28 5.4E-79
- DOUBLEPRECISION : 1.D12 6.281234567D-1 -3.D-27

2.4 Fonctions

Fonctions couramment utilisées, en version REAL :

- AINT (expr) troncation.
- ANINT (expr) arrondi.
- ABS (expr) valeur absolue.
- AMIN1 (.....) minimum de plusieurs expressions.
- AMAX1 (.....) maximum ..

En DOUBLE PRECISION ces fonctions s'appellent : DINT, DNINT, DABS, DMIN1 et DMAX1.

2.5 La trigonométrie, et cie.

- SQRT (expr) : racine carrée.
- EXP (e1,e2) : e1 à la puissance e2.
- ALOG (expr) : log. népérien
- ALOG10(expr) : log. base 10
- SIN (expr) : sinus
- COS (expr) : cosinus
- TAN (expr) : tangente
- ASIN (expr) : arcsinus
- ACOS (expr) : arccosinus
- ATAN (expr) : arctangente

Il y en a d'autres ... Les équivalents en DOUBLEPRECISION s'appellent respectivement : DSQRT, DEXP, DLOG, DLOG10, DSIN, DCOS,DTAN, DASIN, DACOS, et DATAN.

2.6 Conversions explicites de type

Le tableau suivant montre l'ensemble des fonctions de conversion de type.

		type du résultat				
		int	int*4	real	dble	
type arg.	int		IDBL	FLOAT	DFLOAT	
	int*4	ISINGL		FLOAT4	DFLT4	
	real	IFIX	IFIX4		DBLE	
	dble	IDINT	IFIXD4	SNGL		

Chapitre 3

Annexe : utilisation des chaînes de caractères

Contrairement à son ancêtre FORTRAN IV, le langage FORTRAN 77 reconnait l'existence du type prédéfini «chaîne de caractères». Ne nous en privons donc pas.

3.1 Déclaration

La taille des chaînes de caractères est fixée lors de la déclaration des objets de type CHARACTER. Par exemple les déclarations :

```
CHARACTER*10 TRUC
CHARACTER*50  MACHIN
```

fixent la taille des chaines TRUC et MACHIN à 10 et 50 caractères. La taille maximum est 255.

3.2 Constantes

Les constantes peuvent s'écrire sous trois formes :

- Entre quotes : 'Hello ! 'C"est l"heure' . Il faut doubler les quotes internes.
- En notation hexadécimale : Z'1BFF000A' (pratique pour la manipulation des caractères non imprimables ESC,LF,BELL,CR,BS, etc...)
- En notation Hollerith : 7Hl'heure . Dans cette notation, on représente une chaine par un entier (la longueur de la chaine, ici 7) suivi par la lettre H, puis le contenu. Cette notation antique et désuète oblige le malheureux programmeur à compter les caractères, ce qui est particulièrement désagréable.

3.3 Opérations sur les chaines de caractères

3.3.1 Concaténation, affectation, sélection

Exemple :

```
 CHARACTER*10 A,B
 CHARACTER*20 C,D
...
C = A
B = D
C = A // B
D(2:11) = C(1:5) // A(3:7)
```

La première instruction copie le contenu de A dans C. Cétant plus longue, les caractères de droite de C sont remplis avec des blancs (cadrage à gauche).

La seconde instruction copie les premiers caractères de D dans B (troncation).

La troisième instruction met bout-à-bout (concatène) les chaînes A et B, le résultat étant transféré ensuite dans C.

La quatrième instruction concatène les 5 premiers caractères de C avec les caractères 3 à 7 de A, le résultat étant placé dans les caractères 2 à 11 de D (les autres caractères de D ne sont pas touchés par l'opération).

On peut comparer des chaines grâce aux opérateurs habituels : .LT. .LE. .EQ. .NE. .GE. .GT. . Pour faire la comparaison la plus courte des chaines est allongée (virtuellement) par des blancs à droite.

3.3.2 Quelques fonctions

- LEN retourne la longueur d'une chaine.
- CHAR convertit un CHARACTER*1 en INTEGER,
- ICHAR fait l'inverse.
- INDEX (ch1,ch2) retourne un entier qui est la position de départ (à l'intérieur de ch1) d'une chaîne identique à ch2. (0 si ch2 n'apparait pas dans ch1).

3.4 Entrées-sorties

Enfin, rappelons que les lectures-écritures de chaînes se font selon la spécification de format A.

Exemple :

```
      WRITE (*,100) 'LITTLE','VERY_BIG_STRING'
100   FORMAT (2A8)
```

provoque l'écriture de : LITTLE_VERY_BIG

Chapitre 4

Annexe : passage de Paramètres

La communication entre unités de programmes (programme principal,sous-programmes, fonctions) peut se faire de deux manières très différentes : par passage de paramètres ou par partage d'une zone commune.

4.1 Passage de paramètres

Exemple (trivial) :

```
PROGRAM  PP
 INTEGER  A,B,C,D
 READ  (*,*) A,B
 CALL  SP(A,B,C)
 CALL  SP(1,2,D)
 WRITE (*,*) A,B,C,D
 STOP
END

SUBROUTINE   SP(X,Y,Z)
 INTEGER  X,Y,Z
 Z = X + Y
 RETURN
END
```

On parle de *paramètres formels* quand on se place du coté de l'unité de programme appellée (X,Y et Z sont les paramètres formels de SP), et de *paramètres effectifs* d'un appel (A,B,C paramètres effectifs du premier appel à SP dans PP, 1,2,D paramètres effectifs du second).

La première chose à savoir est que les paramètres sont toujours passés *par adresse*. Ainsi, il est possible de modifier - depuis l'intérieur d'un sous-programme - le contenu d'une variable reçue comme paramètre (dans notre exemple, il est clair que

le sous-programme SP modifie son troisième argument). Les résultats d'expressions arithmétiques (et les constantes) utilisés comme paramètres effectifs sont traités de la même manière : dans l'appel CALL SP(A+B,0,C) le premier paramètre effectif sera en fait une variable intermédiaire contenant le résultat de l'addition de A et B, le second paramètre une zone mémoire contenant la valeur 0.

La seconde chose -très importante- à savoir est que la correspondance de types entre arguments effectifs et arguments formels n'est pas vérifiée par le compilateur FORTRAN. Ainsi l'appel CALL SP (1.0,1.0,C) conduira-t-il à un résultat douteux. Il n'y a pas non plus de contrôle du nombre de paramètres. Ces vérifications sont laissées à la charge du programmeur. Rappelons en particulier que les constantes entières entre -32768 et 32767 sont de type INTEGER, et que les constantes plus grandes sont en INTEGER*4.

4.2 Passage de fonctions ou sous-programmes en paramètres

Exemple :

```
            REAL FUNCTION  DERIV(F,X)
C
C Calcule la valeur de la dérivée de la fonction F en X
C
            REAL F,X
              DERIV =  ( F(X+0.01) - F(X-0.01) ) / 0.02
            RETURN
            END

            REAL FUNCTION  TEST (X)
C
C Un exemple de fonction à une variable réelle
C
             TEST = X**2 + SIN(X)
            RETURN
            END

            PROGRAM  ESSAI
            EXTERNAL TEST
            REAL  TEST,POINT
              READ  (*,*) POINT
              WRITE  (*,*) 'Au point ',POINT,
     S                     'la derivee vaut',DERIV(TEST,POINT)
            STOP
            END
```

La déclaration EXTERNAL TEST (du programme principal) signale au compilateur FORTRAN que le nom TEST désigne une unité de programme (et non pas une variable).

Dans DERIV, il est «conseillé» de déclarer le type du paramètre formel F. Le compilateur arrive à déterminer que F est une FUNCTION, car le nom F est suivi de parenthèses, et aucun tableau F n'est déclaré.

Si, dans le programme principal, on avait voulu passer en paramètre une fonction prédéfinie (par exemple SIN), alors on doit utiliser l'instruction INTRINSIC SIN (au lieu d'EXTERNAL).

4.3 Tableaux Formels

Exemple :

```
      REAL FUNCTION SOMME(T,N)
      INTEGER N
      REAL T(N)
C
C Calcul de la somme des N elements du tableau T
C
        INTEGER  I
        REAL  S
C
        S = 0.0
        DO  100,I=1,N
            S = S + T(I)
100     CONTINUE
        SOMME = S
        RETURN
      END
```

Dans cet exemple le tableau (formel) T possède une dimension (formelle) donnée par le paramètre (formel) N. C'est le seul cas de taille formelle possible, car T n'est qu'une référence à un objet existant par ailleurs (dans l'unité appelante), et la déclaration de T ne fait aucune réservation de place mémoire.

4.4 Adresses de retour alternatives

Exemple :

```
          PROGRAM  TRUC
            REAL  X,Y
            READ  (*,*) X,Y,F
            Y = F(X,*99)
            WRITE  (*,*) 'Resultat =',Y
            STOP
 C
C En cas d'erreur ..............................
 99         WRITE  (*,*) 'F n''est pas definie en ',X
            STOP
```

```
 END

REAL FUNCTION F ( X , * )
REAL X,DENOM
 DENOM = X - 1.0
 IF  (DENOM .EQ. 0.0)  RETURN 1
    F = SIN(X) / DENOM
RETURN
END
```

Ici le second paramètre de la fonction F est une adresse de retour alternative. Si l'instruction «RETURN 1» est exécutée dans F, alors un branchement est effectué à l'instruction 99 du programme principal. On peut avoir plusieurs adresses de retour alternatives, c'est la valeur de l'expression située derrière le RETURN qui détermine le numéro de l'adresse choisie.

Dans l'unité appelante, n'oubliez pas de mettre une étoile devant les étiquettes que vous passez en paramètres...

Cette facilité est utilisée très souvent pour indiquer (comme ici) l'adresse d'une instruction à exécuter en cas d'anomalie. Elle est finalement très voisine de la forme «READ(....END=...ERR=...)».

Chapitre 5

Annexe : COMMON

La définition de zones communes permet le partage d'un «pool» de variables entre plusieurs unités de programmes.

Exemple :

```
INTEGER FUNCTION  DEPILE()
COMMON  /PILE/ H,T
INTEGER H,T(10)
  DEPILE = T(H)
  H = H - 1
RETURN
END

SUBROUTINE  EMPILE(X)
COMMON  /PILE/ H,T
INTEGER  H,T(10),X
  H = H + 1
  T(H) = X
RETURN
END
```

Dans ces deux unités l'instruction COMMON déclare les variables H et T comme faisant partie de la zone commune (globale) nommée PILE.

L'intention du programmeur est ici de définir un objet PILE, accessible par le biais de «primitives» : EMPILE, DEPILE, VIDER, etc ... L'usage des COMMONs permet donc d'éviter - dans certains cas - des passages de paramètres inutiles. N'en abusez pas. Dans le cas où, par exemple, un programme doit manipuler plusieurs piles, on préférera passer la pile (hauteur + vecteur) en paramètre plutôt que définir une multitude de zones communes. A cet égard, la philosophie personnelle de l'auteur est la suivante:

On définit un COMMON chaque fois que l'on isole -dans un programme- une entité unique destinée à être utilisé à partir de plusieurs unités de programme. Le COMMON contient les variables propres à cette entité, et l'accès à ces variables ne doit sefaire qu'à partir d'un nombre restreint de primitives d'accès.

Attention à l'ordre et au type des éléments d'un COMMON, le compilateur FORTRAN ne vérifie pas la cohérence de type d'une unité à l'autre.... En pratique il est très fortement conseillé de reproduire textuellement les instructions COMMON .

Pour finir, notez que l'initialisation statique (par DATA) des variables d'un COMMON ne peut se faire qu'à l'intérieur d'une unité de programme BLOCK DATA (qui, d'ailleurs, ne sert qu'à çà).

Exemple : Si on veut initialiser la pile à vide :

```
BLOCK DATA INIT
COMMON /PILE/ H,T
INTEGER H,T(10)
DATA H / 0 /
END
```

Chapitre 6

Annexe : entrées-Sorties directes formattées

Un fichier en accès direct peut être assimilé (approximativement) à un tableau d'enregistrements. L'accès (lecture ou écriture) se fait donc toujours en précisant le numéro d'enregistrement (un entier positif).

Exemple :

```
            PROGRAM  DEMOAD
C
C Demonstration de l'acces direct.
C
C le fichier AD.DAT contient des enregistrement de 20 caractères
C l'utilisateur peut Voir ou Modifier chaque enregistrement.
C
            CHARACTER*1  COMMANDE
            CHARACTER*20 ENREG

            INTEGER  NUMERO
C
C
C Ouverture du fichier AD.DAT en accès direct.
C
            OPEN  ( 1 , FILE = 'AD.DAT',
     S              ACCESS = 'DIRECT',
     S                FORM = 'FORMATTED'
     S                RECL = 20          )
C
C On demande la commande ...
C
1000        WRITE  (*,*) 'V(oir), M(odifier), S(top)'
            READ (*,1010) COMMANDE
C
C On regarde la commande ...
C
```

```
           IF  (COMMANDE.EQ.'V')  THEN
C
C .... Voir ...
C
                WRITE  (*,*) 'Voir quel numéro ?'
                READ  (*,*) NUMERO
                READ (1,100,REC=NUMERO,ERR=1100) ENREG
100             FORMAT  (A20)
                WRITE  (*,*) 'Enreg',NUMERO,'=',ENREG
                GOTO  1000
C
1100            WRITE  (*,*) 'Il n''y a pas d''enregistrement',
     S                           'numero',NUMERO
                GOTO 1000
C
            ELSE IF (COMMANDE.EQ.'M')  THEN
C
C .... Modifier ...
C
                WRITE  (*,*) 'Modifier quel numéro ?'
                READ  (*,*) NUMERO
                WRITE  (*,*) 'Mettre quoi à la place ?'
                READ  (*,100) ENREG
                WRITE (1,100,REC=NUMERO,ERR=1200) ENREG
                GOTO  1000
C
1200            WRITE  (*,*) 'On ne peut pas modifer',
     S                  'l''enregistrement numero',NUMERO
                GOTO  1000
C
C           ELSE IF  (COMMANDE.EQ.'S')  THEN
C
C .... stop ...
C
C               CLOSE  (1)
C               STOP  'Ok.'
C
            ELSE
C
C .... autre commande ...
C
                WRITE  (*,*) 'Je ne comprends que V,C ou S.'
                GOTO  1000
C
C           ENDIF
C
            END
```

Pour l'ouverture d'un fichier en accès direct, il faut obligatoirement donner les clauses « ACCESS='DIRECT',FORM='FORMATTED' » et indiquer la longueur de l'enregistrement (en octets) par « RECL=... » qui doit correspondre à la longueur

des formats utilisés avec ce fichier.

Dans les ordres READ et WRITE, la clause «REC=...» permet de préciser le numéro de l'enregistrement que l'on veutlire ou écrire. Attention, toute tentative de lecture d'unenregistrement absent se solde par une erreur, que l'on peut (heureusement) récupérer par l'option «ERR=...» .