TABLE DES MATIERES

INTRODUCTION

OBJECTIFS D'UNE NORMALISATION DU FORTRAN-77

MODULAD a pour vocation principale de développer des applications en Analyse des Données qui soient aisément portables d'une machine à l'autre, quel que soit l'environnement. Or, fruit d'une longue expérience, il est toujours apparu nécessaire d'imposer des restrictions supplémentaires dans l'usage des langages de programmation vis-à-vis des normes internationales.

Le premier objectif de ces normes MODULAD est de proposer à tout développeur un document de référence lui permettant d'écrire des programmes en FORTRAN-77 avec le quasi certitude que, s'il respecte les recommandations proposées, il ne rencontrera aucun problème de portabilité de ses sources.

Un autre objectif de ce document est d'inciter chaque développeur à adopter systématiquement une écriture structurée pour la rédaction de tout logiciel, dans la quadruple perspective de faciliter les problèmes d'écriture, de relecture, de mise au point et d'évolution.

Ces normes, proposées pour le développement d'applications en FORTRAN-77, dépassent très largement le seul cadre de MODULAD. Elles pourront être mises en oeuvre avec profit dans bien d'autres domaines que l'Analyse des Données.

UNE NOUVELLE VERSION DU DOCUMENT "NORMALISATION FORTRAN-77"

Pourquoi proposer, dans le cadre de MODULAD, un nouvel ensemble de normes pour la rédaction des programmes écrits en FORTRAN-77? Plusieurs justifications peuvent être avancées.

Tout d'abord, les compilateurs disponibles dans le monde de la micro-informatique comme dans celui des grands systèmes ont sensiblement évolué. En particulier, concernant les contraintes qu'ils induisaient dans les premières normes proposées par MODULAD pour l'écriture en FORTRAN-77, beaucoup de restrictions imposées eu égard à la norme 77 peuvent être aujourd'hui supprimées.

D'autre part, il était nécessaire de corriger quelques incohérences que contenait le document précédent et de modifier dans un sens parfois plus restrictif l'utilisation de certaines instructions.

Enfin, et c'est peut-être le point majeur, il était devenu indispensable d'intégrer dans ce document les nouvelles normes de programmation FORTRAN-77 introduites par IBM dans le cadre de l'Architecture Unifiée d'Applications (SAA). Le document SAA/CPI/FORTRAN-77 constitue aujourd'hui une référence difficilement contournable si l'on souhaite développer des applications portables, ce qui est l'objectif primordial des normes proposées par MODULAD.

1-ère partie

DEFINITION D'UN SOUS-ENSEMBLE FORTRAN-77 PORTABLE

1-1 LES ELEMENTS UTILISABLES EN FORTRAN-77

Dans ce paragraphe, nous précisons quelles sont les règles de codification des différents éléments intervenant dans les instructions (constantes, variables, ...), ainsi que la manière d'écrire ces instructions.

1-1.1 *JEU DES CARACTERES UTILISABLES*

autorisés : les caractères alphabétiques majuscules : A B C ... Z
les caractères alphabétiques minuscules non accentués : a b c ... z (*hors norme 77*)
les caractères numériques : 0 1 2 ... 9

les caractères spéciaux suivants :

	blanc	(space)
_	blanc souligné	(underscore) (*hors norme 77*)
,	virgule	(comma)
.	point décimal	(period)
:	deux points	(colon)
'	apostophe	(single quotation)
(	parenthèse ouvrante	(left parenthesis)
)	parenthèse fermante	(right parenthesis)
=	signe d'égalité	(equal sign)
+	plus	(plus sign)
-	moins	(minus sign)
*	étoile	(asterisk)
/	barre de fraction	(slash)

interdits : les caractères suivants : (cas particuliers : commentaires et chaînes)

"	guillemet	(double quotation)
;	point virgule	(semi-colon)
?	point d'interrogation	(question mark)
!	point d'exclamation	(exclamation mark)
$	symbole dollar	(currency symbol)
&	et commercial	(ampersand)
<	plus petit que	(left angle bracket)
>	plus grand que	(right angle bracket)

tout caractère de tabulation
tout caractère de code ASCII plus petit que 32 ou plus grand que 127

De façon générale, est interdit tout caractère ne figurant pas parmi ceux autorisés (en particulier, est exclu tout caractère accentué ou semi-graphique).

remarque : La norme 77 impose de n'utiliser que des majuscules. Cependant, de plus en plus de compilateurs acceptent les minuscules non accentuées, ce qui nous conduit à les autoriser. Toutefois, veillez à ne pas mélanger majuscules et minuscules dans les instructions (sauf pour les constantes chaînes de caractères ou dans les commentaires).

1-1.2 *DISPOSITION DES INSTRUCTIONS*

Les instructions sont saisies n'importe où entre les colonnes 7 à 72. On ne doit placer qu'une seule instruction par ligne. En particulier, il est interdit d'utiliser le caractère point-virgule (;) pour séparer plusieurs instructions sur une même ligne.

Une instruction peut s'étendre sur plusieurs lignes à condition qu'un caractère, autre que blanc ou zéro, apparaisse en colonne 6. Nous recommandons d utiliser le symbole plus (+) ou le symbole (S) pour indiquer une telle *continuation*, en se limitant à 9 lignes au maximum. On ne doit pas disposer, à cheval entre deux lignes, un mot-clé du langage, un nom de variable ou une constante (qu'elle soit de type numérique ou chaîne de caractères).

Il ne faut pas placer de *ligne blanche* (vide) dans le flot des instructions.

Dans les colonnes 1 à 5, les *étiquettes* sont obligatoirement cadrées à droite de la zone qui leur est réservée (cadrage sur la colonne 5).

Il ne faut jamais utiliser la colonne 1, sauf pour le C ou l'astérisque (*) des lignes de commentaire. De même, les *colonnes 73 à 80* doivent rester inutilisées : elles peuvent servir à stocker des informations MODULAD pour la portabilité des instructions d'une machine à une autre; ceci est également vrai pour les lignes de commentaire.

La saisie des instructions sous *forme libre* (possibilité offerte avec la norme 77) est interdite

Utilisation du caractère blanc comme séparateur dans les instructions :

Dans toutes les instructions, les étiquettes, les mots-clés, les noms d'identificateurs (y compris les noms de sous-programmes) et les constantes doivent obligatoirement *être séparés par au moins un séparateur* (caractère spécial ou blanc).

cas particulier : laisser au moins un caractère blanc derrière une parenthèse fermante dans les cas suivants :

- derrière la dernière parenthèse fermante de l'expression logique d'un IF logique.
- derrière la première parenthèse fermante dans une instruction d'entrée/sortie READ ou WRITE.

1-1.3 *COMMENTAIRES*

Pour introduire un commentaire, il suffit de placer un caractère (C) ou astérisque (*) en début de ligne (colonne 1). Les colonnes 2 à 72 sont alors entièrement disponibles pour un commentaire (ne pas utiliser les colonnes 73 à 80 réservées MODULAD). Plusieurs lignes de commentaires peuvent se suivre. Elles peuvent être placées n'importe où dans une entité de programme, y compris avant la première instruction, mais *jamais après la dernière instruction END d'un fichier source*.

S'il est interdit d'avoir des lignes entièrement blanches (vides), il est parfaitement autorisé d'introduire des lignes de commentaire ne contenant qu'un caractère (C ou *) est colonne 1 (ligne de commentaire vide).

Il ne faut jamais utiliser la possibilité, parfois offerte, d'introduire des commentaires à la fin d'une ligne contenant une instruction.

Dans les commentaires, il est possible d'introduire certains des caractères interdits pourvu qu'ils correspondent, dans la table des codes ASCII, à des valeurs décimales comprises entre 32 et 127 (les caractères accentués et les symboles semi-graphiques sont exclus).

1-1.4 *ORDRE D'APPARITION DES INSTRUCTIONS*

L'ordre suivant doit *obligatoirement* être respecté :

(1)	PROGRAM, SUBROUTINE ou FUNCTION
(2)	spécification explicite de type pour les constantes PARAMETER
(3)	PARAMETER
(4)	spécification explicite de type pour les variables
(5)	SAVE
(6)	COMMON
(7)	instructions exécutables
(8)	END

Les instructions FORMAT peuvent être placées n'importe où parmi les instructions exécutables. Il est toutefois conseillé de les regrouper à la fin des instructions exécutables, juste avant l'instruction END.

Les commentaires peuvent être placés n'importe où, y compris avant (1), mais pas après (8).

1-1.5 *CONSTANTES*

autorisées : - *constantes numériques :*

* les entiers sont compris entre -2147483648 (-2**31) et +2147483647 (2**31-1); il s'agit de la représentation habituelle des entiers, sour forme complément à 2 sur 4 octets.

* les réels doivent être compris, en valeur absolue, entre 1E+35 et 1E-35; il s'agit d'une limite "basse" autorisant un compromis entre les divers types de représentations internes de nombre réels simple ou double précision.

* la précision des réels "simple précision" est limitée à 6 chiffres décimaux significatifs, et celle des réels "double précision" à 15 chiffres; il s'agit, là aussi, d'une limite "basse" prenant en compte les divers types de représentations internes.

- *constantes logiques :*

* .TRUE. et .FALSE.

- *constantes chaînes de caractères :*

* toute séquence de caractères délimitée par des caractères apostrophes ('...'); pour des questions de portabilité, on se limitera au caractères disponibles, dans le code ASCII, entre 32 et 127 (sont exclus tous les caractères accentués, ainsi que tous les symboles semi-graphiques).

* les constantes Hollerith (nH...) sont utilisables, mais *uniquement dans les formats des instructions de sortie impression*, comme caractères de contrôle du saut du papier.

- *constantes complexes :*

* sous la forme : (partie-réelle,partie-imaginaire)

interdites :
- constantes octales ou hexadécimales
- .T. ou .F. pour coder les constantes logiques (écrire .TRUE. ou .FALSE.)
- les chaînes de caractères définies entre caractères guillemets ("....")
- les chaînes de caractères contenant des caractères autres que ceux compris, dans le code ASCII, entre 32 et 127
- les constantes chaînes de caractères Hollerith (nH....) si elles ne figurent pas dans un format lié à une instruction d'entrée/sortie
- de façon générale, tout type de constante autre que ceux autorisés

1-1.6 *NOMS DES IDENTIFICATEURS*

autorisés :
- *nom d'entité de programme* : *(PROGRAM, SUBROUTINE et FUNCTION)*
  * de *1 à 6 caractères* alphanumériques, le premier étant obligatoirement alphabétique
- *nom de COMMON* :
  * de *1 à 6 caractères* alphanumériques, le premier étant obligatoirement alphabétique
- *nom de variables* : *(variable simple, tableau et constante PARAMETER)*
  * de *1 à 31 caractères* alphanumériques ou blanc souligné (_), le premier étant obligatoirement alphabétique (*hors norme 77*)

interdits :
- tous les mots-clés réservés du langage FORTRAN-77 (y compris les mots-clés liés aux instructions interdites dans le cadre de MODULAD)
- tous les noms de fonctions de la bibliothèque FORTRAN-77
- l'utilisation du caractère dollar ($) dans les noms d'identificateurs
- de façon générale, est interdit tout nom ne respectant pas les règles autorisées

remarque :
Le fait d'autoriser des noms de variables dépassant 6 caractères et admettant des caractères blanc souligné (_) n'est pas conforme à la norme 77. Cependant, nombreux sont les compilateurs autorisant jusqu'à 31 caractères (avec blanc souligné); aussi, nous a-t-il paru souhaitable de ne pas se priver de cette possibilité, toujours plus répandue, dans le cadre des développements MODULAD.

Le respect de la norme 77 impose de ne pas dépasser 6 caractères, avec interdiction du blanc souligné. Il est toujours possible de continuer à s'imposer cette règle très restrictive dans le cadre de MODULAD, afin d'assurer une portabilité vraiment complète.

Cette règle, avec limitation à 6 caractères sans blanc souligné, doit *impérativement* être respectée pour les noms d'entités de programmes et de COMMON nommés.

1-1.7 *ETIQUETTES*

Normaliser des étiquettes consiste à *numéroter des étiquettes, en ordre croissant d'apparition, dans la zone des colonnes 2 à 5*, en distinguant les étiquettes des instructions exécutables de celles référençant des instructions FORMAT. L'usage d'étiquettes doit être le plus limité possible, sinon la structure logique du programme risque d'apparaître de façon beaucoup moins évidente. Le recours à une plus grande modularité (éclatement en sous-programmes) favorise une économie d'étiquettes.

En préfixe devant une instruction, les étiquettes doivent être saisies en *colonne 2 à 5* et cadrées à droite dans cette zone (c'est-à-dire *alignées en colonne 5*).

1-1.7.1 *Etiquettes devant des instructions exécutables*

Ne sont pas prises en compte ici les étiquettes précédant une instruction FORMAT.

Placées dans les colonnes 2 à 5, ce type d'étiquette ne doit *jamais se rencontrer ailleurs que devant une instruction CONTINUE*.

autorisées : les étiquettes, incrémentées de 10 en 10, depuis la première étiquette 10 jusqu'à l'étiquette 990 (soit 99 étiquettes différentes disponibles dans une entité de programme.

interdites :
- les étiquettes non disposées en ordre croissant
- les étiquettes non incrémentées de 10 en 10
- les étiquettes supérieures à 990
- de façon générale, toute étiquette autre que celles autorisées.

1-1.7.2 *Etiquettes devant des instructions FORMAT*

Là encore, les étiquettes doivent être *incrémentées de 10 en 10, avec une distinction des instructions FORMAT selon les types d'entrée/sortie*.

autorisées :
- *format d'édition* :
  * les étiquettes vont de 1000 à 1990.
- *format de lecture* :
  * les étiquettes vont de 2000 à 2990.
- *format d'entré/sortie sur supports magétiques* :
  * les étiquettes vont de 3000 à 3990.

interdites : toute étiquette de FORMAT non conforme à la fois aux normes d'incrémentation de 10 en 10 ou à la caractérisation du type d'entrée/sortie.

remarque : Le nombre d'instructions FORMAT doit être réduit au minimum, car il faut, bien sûr, leur préférer des *formats directement décrits dans les ordres d'entrée/sortie*.

1-2 *LES INSTRUCTIONS FORTRAN-77*

Nous présentons, dans ce paragraphe, les différentes instructions FORTRAN-77, en précisant bien lesquelles sont autorisées dans le cadre de MODULAD.

1-2.1 *INSTRUCTIONS DEFINISSANT UNE ENTITE DE PROGRAMME*

autorisées : PROGRAM

SUBROUTINE
type FUNCTION (où type=INTEGER, REAL, LOGICAL, ...)

interdites : ENTRY

FUNCTION (utiliser *systématiquement* un type explicite pour les fonctions)

instruction fonction

remarque : Il faut bien évidemment interdire ENTRY, car il s'agit d'une logique de programmation allant à l'encontre des principes de la programmation structurée.

1-2.2 *INSTRUCTIONS DE SPECIFICATION (non exécutables)*

autorisées : INTEGER (doit être l'équivalent d'un "entier long")
REAL
DOUBLE PRECISION
COMPLEX
LOGICAL
CHARACTER*n (n ne doit pas dépasser 255)

COMMON avec nom

EXTERNAL
INTRINSIC

SAVE (mais avec usage restrictif)

interdites : NAMELIST

IMPLICIT
IMPLICIT NONE

INTEGER*n
REAL*n
LOGICAL*n
CHARACTER (utiliser CHARACTER*1)
COMPLEX*n
DOUBLE COMPLEX
BYTE
FIXED
DOUBLE FIXED
FRACTIONNAL
DOUBLE FRACTIONNAL

INTERFACE TO

interdites (suite) : DIMENSION (utiliser des déclarations avec type explicite)

COMMON blanc

EQUIVALENCE

AUTOMATIC

MAP
END MAP

STRUCTURE
END STRUCTURE

UNION
END UNION

RECORD

De façon générale, est interdite toute instruction de déclaration non autorisée.

remarques : *Tous les identificateurs*, sans exception, *doivent être déclarés* (en particulier toutes les variables et les tableaux). Il s'agit de prendre modèle sur la plupart des autres langages de programmation, tendance que l'on commence également à percevoir en FORTRAN : il existe d'ailleurs sur beaucoup de compilateurs une option de compilation permettant de détecter les variables non déclarées...

Conséquence de l'obligation de déclaration de tout identificateur, il faut interdire l'utilisation de l'instruction IMPLICIT, liée aux déclarations par défaut.

L'instruction DIMENSION est à rejeter, il faut déclarer explicitement le type des tableaux. Ceci est cohérent avec l'interdiction d'utiliser l'instruction IMPLICIT.

Pour les entiers, certains compilateurs utilisent encore par défaut des déclarations de longueur 2 octets au lieu de 4. Dans ce cas, et en l'absence de toute option de compilation particulière qui permettrait de changer la longueur par défaut des entiers déclarés INTEGER, il est alors possible d'utiliser le type INTEGER*4.

Les variables chaines de caractères sont limitées à 255 caractères.

Il faut utiliser le moins possible la déclaration INTRINSIC, car elle pose quelques problèmes avec les noms de fonctions génériques de la bibliothèque FORTRAN : en effet, ces fonctions perdent alors leur propriété "générique".

1-2.3 *INSTRUCTIONS D'INITIALISATION (non exécutables)*

autorisée : PARAMETER

interdites : DATA (à remplacer par des affectations)
BLOCK DATA (à remplacer par des affectations)

remarque : Une initialisation de type DATA, que l'on trouve également dans les entités BLOCK DATA, présente l'inconvénient d'être une initialisation à caractère statique (prise en compte une seule fois au moment du chargement d'un programme) : conséquence, une instruction DATA, dans un sous-programme appelé plusieurs fois, n'a aucun effet, ce qui peut conduire à des anomalies dans le déroulement de ce sous-programme...

Il est toujours possible et guère plus "coûteux" de *remplacer* tous les DATA et BLOCK DATA *par des affectations*, ce qui est de toute façon plus clair.

Bien évidemment, sont exclus toutes les initialisations de "type DATA" que l'on pourrait rencontrer dans un certain nombre de déclarations de type, voire de COMMON.

1-2.4 *INSTRUCTIONS DE CONTROLE (exécutables)*

autorisées : = (affectation)

ASSIGN ... TO ... (affectation d'une constante étiquette à une variable)

DO ...
CONTINUE

IF logique

IF ... THEN
ELSE
ELSEIF ... THEN
ENDIF

GOTO inconditionnel (mais avec usage très restrictif)

CALL
RETURN
RETURN n (mais avec usage très restrictif)

STOP (a priori inutile)
END

interdites : GOTO calculé
GOTO assigné

IF arithmétique

DO WHILE ...
ENDDO
WHILE ...
EXIT
CYCLE

SELECT CASE ...
CASE ...
CASE DEFAULT
END SELECT

WAIT
PAUSE

CHAIN

ALLOCATE
DEALLOCATE

De façon générale, est interdite toute instruction exécutable non autorisée.

remarques : Les instructions GOTO assigné et GOTO calculé vont à l'encontre des règles de la programmation structurée.

L'instruction ASSIGN ne doit servir que pour ranger des étiquettes constantes de FORMAT dans des variables (étiquettes variables pour les formats dans les entrées/sorties).

L'instruction GOTO inconditionnel doit être d'un usage complètement banilisé et *ne jamais renvoyer sur autre chose qu'une instruction CONTINUE*. Elle ne doit servir que dans le cadre des "figures" de programmation structurée qui sont exposées dans la partie "Règles et conseils de programmation".

L'*instruction DO de boucle avec index* doit continuer d'être utilisée *avec une étiquette*.

L'instruction RETURN n ne doit être utilisée que dans le cas très précis de la mise en oeuvre d'une gestion des erreurs, pour faciliter les retours de sous-programmes en cascade.

L'instruction STOP ne devrait pas être nécéssaire dans un programme.

Les instructions de programmation structurée telles que DO WHILE ... ENDDO ou SELECT CASE ... END SELECT ne font hélas pas partie de la norme 77.

Les instructions ALLOCATE et DEALLOCATE correspondent à des allocations dynamiques de tableaux, mais ne font pas non plus partie de la norme 77 (*patience...*).

1-2.5 *INSTRUCTIONS D'ENTREE/SORTIE (exécutables)*

Les instructions d'entrée/sortie sont comme dans tout langage celles qui posent le plus de problèmes de portabilité.

1-2.5.1 *Liste des instructions*

autorisées :

OPEN
CLOSE

READ
WRITE

REWIND

INQUIRE

interdites :

PRINT
PUNCH
TYPE
ACCEPT
DISPLAY

READ avec NAMELIST
WRITE avec NAMELIST

BACKSPACE
ENDFILE

DEFINE FILE
FIND

REWRITE

LOCKING
UNLOCK

DELETE

De façon générale, est interdite toute instruction d'entrée/sortie non mentionnée comme autorisée.

remarque : La plupart des instructions d'entrée/sortie autorisées comportent néanmoins des restrictions d'utilisation qui sont détaillées dans les paragraphes suivants.

1-2.5.2 *Paramètres des instructions d'entrée/sortie*

autorisés :

UNIT=unité
ERR=étiqerr
IOSTAT=ios

FMT=format

autorisés : END=étiqfin
(suite)

FILE=nomfich
ACCESS=accès
FORM=form
BLANK=typblanc
STATUS=statut (mais avec des restrictions importantes)

RECL=lenr
REC=ienr

DIRECT= (pour ces paramètres, voir INQUIRE)
EXIST=
FORMATTED=
NAME=
NAMED=
NEXTREC=
NUMBER=
OPENED=
SEQUENTIAL=
UNFORMATTED=

interdits : ACTION=
BINARY=
BLOCKSIZE=
DUPKEY=
ID=
KEY=
KEYED=
KEYEND=
KEYGE=
KEYGT=
KEYID=
KEYLENGTH=
KEYS=
KEYSTART=
LASTKEY=
LASTREC=
LOCKMODE=
MODE=
NML=
NOTFOUND=
NUM=
PASSWORD=
READ=
READWRITE=
RECORDS=
SHARE=
WRITE=

De façon générale, sont interdits tous les paramètres ne figurant pas dans la liste de ceux autorisés.

remarque : Le fait de pouvoir utiliser un paramètre d'instruction d'entrée/sortie n'implique pas que toutes les codifications de valeurs admises pour ce paramètre soient autorisées.

1-2.5.3 *Codification des paramètres des instructions d'entrée/sortie*

unité : variable entière contenant un numéro d'unité logique (pas de constante, pas d'expression). Ce numéro d'unité logique doit normalement être associé à un fichier, sauf éventuellement dans une instruction INQUIRE.
Ne pas pas utiliser de numéro d'unité logique supérieur à 99 et ne pas utiliser les numéros 1 à 9, sauf pour des entrées/sorties de type console ou impression.
L'utilisation du caractères astérisque (*) comme unité logique est interdite.

nomfich : une constante, une variable ou une expression de type chaîne de caractères, contenant le nom de l'ensemble de données qui sera associé à un numéro d'unité logique au moment d'une instruction OPEN ou bien qui servira pour une interrogation INQUIRE. La longueur maximum à prévoir pour cette chaîne est de 80 caractères.

format : une constante étiquette renvoyant à une instruction FORMAT, une variable entière contenant une constante étiquette renvoyant à une instruction FORMAT (affectation par ASSIGN uniquement), une constante, une variable, un tableau ou une expression de type chaîne de caractères décrivant un format.

étiqfin : une constante étiquette indiquant où se débrancher en cas de fin de fichier (uniquement en lecture et pour des fichiers séquentiels).

étiqerr : une constante étiquette indiquant où se débrancher en cas d'erreur dans l'exécution d'une instruction d'entrée/sortie.

ios : variable entière permettant éventuellement de vérifier si une instruction d'entrée/sortie s'est déroulée sans anomalie. Après une instruction d'entrée/sortie, cette variable contiendra 0 si l'instruction s'est correctement exécutée, et une valeur différente de 0 sinon.

accès : constante, variable ou expression de type chaîne de caractères ne pouvant contenir que 'DIRECT' ou 'SEQUENTIAL' (par défaut 'SEQUENTIAL').

form : constante, variable ou expression de type chaîne de caractères ne pouvant contenir que 'FORMATTED' ou 'UNFORMATTED'.
Si accès='SEQUENTIAL', alors par défaut form='FORMATTED'.
Si accès='DIRECT', alors par défaut form='UNFORMATTED'.

lenr : constante, variable ou expression entière permettant de préciser la longueur d'un enregistrement (*en nombre d'octets*) lors d'une instruction OPEN concernant l'ouverture d'un fichier à accès direct. La longueur d'un enregistrement en accès direct est obligatoirement fixe.
Si le fichier est ouvert avec form='FORMATTED', *lenr* correspond alors au nombre maximum de caractères transmis qui est fonction des différents formats utilisés.
Si le fichier est ouvert avec form='UNFORMATTED', *lenr* correspond alors au nombre maximum d'octets transmis lié à la représentation interne des données (dans ce cas, il faut faire attention à la portabilité de la représentation interne, en particulier pour les entiers).

ienr : contante, variable ou expression entière qui correspond à un numéro d'enregistrement à manipuler pour des entrées/sorties à accès direct. Les numéros d'enregistrements vont de 1 à N.

chaine : variable chaîne de caractères (éventuellement sous-chaîne ou tableau de caractères).

typblanc : constante, variable ou expression de type chaîne de caractères ne pouvant contenir que 'ZERO' ou 'NULL' (par défaut : 'NULL' si un OPEN a été réalisé sur le fichier, mais 'ZERO' si le fichier n'a fait l'objet d'aucun OPEN).

statut : constante, variable ou expression de type chaîne de caractères.

Pour une instruction OPEN, on peut coder 'UNKNOWN' qui est la valeur par défaut. On peut aussi utiliser 'NEW', 'OLD' ou 'SCRATCH', en sachant qu'ils sont sans effet sur certains systèmes.

Pour une instruction CLOSE, on peut coder 'KEEP' ou 'DELETE' (par défaut : 'KEEP' si, dans OPEN, statut était différent de 'SCRATCH', sinon 'DELETE'). Attention, 'KEEP' et 'DELETE' sont sans effet sur certains systèmes.

1-2.5.4 *Instructions d'entrée/sortie de type console ou impression*

autorisées :

```
READ (unité,format)  liste
READ (unité,format,END=étiqfin)  liste
WRITE (unité,format)  [liste]

READ ([UNIT=]unité,[FMT=]format[,END=étiqfin][,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité,[FMT=]format[,ERR=étiqerr][,IOSTAT=ios])  [liste]


READ (unité,*)  liste
READ (unité,*,END=étiqfin)  liste
WRITE (unité,*)  [liste]

READ ([UNIT=]unité,[FMT=]*[,END=étiqfin][,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité,[FMT=]*[,ERR=étiqerr][,IOSTAT=ios])  [liste]
```

interdites : READ format, liste
PRINT format [,liste]
PUNCH format [,liste]

READ *, liste
PRINT *, liste
PUNCH *, liste

READ avec NAMELIST
WRITE avec NAMELIST

ACCEPT
TYPE
DISPLAY

remarque : Le format libre (*) est autorisé avec ce type d'entrée/sortie.

1-2.5.5 *Instructions d'entrée/sortie sur supports magnétiques (accès séquentiels)*

autorisées : - *syntaxe générale*

```
OPEN ([UNIT=]unité[,FILE=nomfich][,ERR=étiqerr][,IOSTAT=ios]
     [,ACCESS='SEQUENTIAL'][,FORM=form][,BLANK=typblanc][,STATUS=statut])

CLOSE ([UNIT=]unité[,ERR=étiqerr][,IOSTAT=ios][,STATUS=statut])

READ ([UNIT=]unité[,[FMT=]format][,END=étiqfin][,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité[,[FMT=]format][,ERR=étiqerr][,IOSTAT=ios])  [liste]

REWIND ([UNIT=]unité[,ERR=étiqerr][,IOSTAT=ios])
```

- *syntaxe particulière pour les entrées/sorties avec format*

```
OPEN ([UNIT=]unité[,FILE=nomfich][,ERR=étiqerr][,IOSTAT=ios]
     [,ACCESS='SEQUENTIAL'][,FORM='FORMATTED'][,BLANK=typblanc][,STATUS=statut])

READ ([UNIT=]unité,[FMT=]format[,END=étiqfin][,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité,[FMT=]format[,ERR=étiqerr][,IOSTAT=ios])  [liste]
```

- *syntaxe particulière pour les entrées/sorties sans format*

```
OPEN ([UNIT=]unité[,FILE=nomfich][,ERR=étiqerr][,IOSTAT=ios]
     [,ACCESS='SEQUENTIAL'][,FORM='UNFORMATTED'][,STATUS=statut])

READ ([UNIT=]unité[,END=étiqfin][,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité[,ERR=étiqerr][,IOSTAT=ios])  [liste]
```

interdites : BACKSPACE
ENDFILE

REWIND unité (utiliser la forme autorisée)

READ (...,[FMT=]*...) liste
WRITE (...,[FMT=]*...) [liste]

Sont interdites toutes les manipulations d'entrée/sortie sur supports magnétiques qui ne débuteraient pas par un OPEN et qui ne se termineraient pas par un CLOSE.

remarque : Le format libre (*) est *interdit* avec ce type d'entrée/sortie.

N'utilisez que des numéros d'unités logiques (UNIT=) compris entre 10 et 99.

Les instructions BACSPACE et ENDFILE ont des comportements parfois très différents selon les systèmes.

1-2.5.6 *Instructions d'entrée/sortie sur supports magnétiques (accès directs)*

autorisées : - *syntaxe générale*

```
OPEN ([UNIT=]unité[,FILE=nomfich][,ERR=étiqerr][,IOSTAT=ios]
     [,ACCESS='DIRECT'][,FORM=form],RECL=lenr[,BLANK=typblanc][,STATUS=statut])

CLOSE ([UNIT=]unité[,ERR=étiqerr][,IOSTAT=ios][,STATUS=statut])

READ ([UNIT=]unité[,[FMT=]format],REC=ienr[,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité[,[FMT=]format],REC=ienr[,ERR=étiqerr][,IOSTAT=ios])  liste
```

- *syntaxe particulière pour les entrées/sorties avec format*

```
OPEN ([UNIT=]unité[,FILE=nomfich][,ERR=étiqerr][,IOSTAT=ios]
     [,ACCESS='DIRECT'][,FORM='FORMATTED'],RECL=lenr[,BLANK=typblanc][,STATUS=statut])

READ ([UNIT=]unité,[FMT=]format,REC=ienr[,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité,[FMT=]format,REC=ienr[,ERR=étiqerr][,IOSTAT=ios])  liste
```

- *syntaxe particulière pour les entrées/sorties sans format*

```
OPEN ([UNIT=]unité[,FILE=nomfich][,ERR=étiqerr][,IOSTAT=ios]
     [,ACCESS='DIRECT'][,FORM='UNFORMATTED'],RECL=lenr[,STATUS=statut])

READ ([UNIT=]unité,REC=ienr[,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE ([UNIT=]unité,REC=ienr[,ERR=étiqerr][,IOSTAT=ios])  liste
```

interdites :

```
DEFINE FILE
FIND
REWRITE

READ (unité ' ienr [,format])  liste
WRITE (unité ' ienr [,format])  liste

READ (...,[FMT=]*,REC=ienr,...)  liste
WRITE (...,[FMT=]*,REC=ienr,...)  liste
```

Sont en particulier interdites toutes les manipulations de type accès direct mettant en jeu des organisations de type "séquentiel indexé" ou "base de données".

remarque : Le format libre (*) est *interdit* avec ce type d'entrée/sortie.

N'utilisez que des numéros d'unités logiques (UNIT=) compris entre 10 et 99.

1-2.5.7 *Instructions d'entrée/sortie avec des fichiers "mémoire centrale"*

Il s'agit de lire et/ou d'écrire des données en mémoire centrale en réalisant des entrées/sorties, (*avec format*) dans des zones de type chaînes de caractères. En particulier, il est possible d'utiliser des formats différents entre lecture et écriture ce qui autorise de nombreuses conversions.

Cette forme d'entrées/sorties est aussi qualifiée de type "fichier interne", par opposition aux "fichiers externes" qui désignent alors les supports magnétiques. Ces manipulations remplacent les anciennes instructions ENCODE et DECODE qui sont désormais interdites.

autorisées :

```
READ (UNIT=chaine,FMT=format,[,ERR=étiqerr][,IOSTAT=ios])  liste
WRITE (UNIT=chaine,FMT=format[,ERR=étiqerr][,IOSTAT=ios])  liste
```

interdites :

```
READ (chaine,format)       (utilisez toujours les mots-clés UNIT et FMT)
WRITE (chaine,format)      (utilisez toujours les mots-clés UNIT et FMT)

READ (...,[FMT=]*)       (pas de format libre avec les fichiers internes)
WRITE (...,[FMT=]*)      (pas de format libre avec les fichiers internes)
```

Pas d'instructions OPEN, CLOSE et REWIND avec les fichiers internes.

1-2.5.8 *Instruction INQUIRE*

Les deux formes de l'instruction INQUIRE sont autorisées (INQUIRE par nom de fichier ou INQUIRE par unité logique).

autorisées :
```
INQUIRE ( { FILE=nomfich | UNIT=unité } [,ERR=étiqerr][,IOSTAT=ios]
          [,EXIST=exs][,OPENED=opn]
          [,NAMED=nmd][,NAME=nam][,NUMBER=num]
          [,ACCESS=acc][,SEQUENTIAL=seq][,DIRECT=dir]
          [,FORM=frm][,FORMATTED=fmt][,UNFORMATTED=unf]
          [,BLANK=blk]
          [,RECL=rcl][,NEXTREC=nxr])
```

exs : avec INQUIRE par nom, variable logique qui contiendra .TRUE. si le nom de fichier existe, .FALSE. sinon.
avec INQUIRE par unité, variable logique qui contiendra .TRUE. si l'unité logique existe, .FALSE. sinon.

opn : avec INQUIRE par nom, variable logique qui contiendra .TRUE. si le fichier est ouvert, .FALSE. sinon.
avec INQUIRE par unité, variable logique qui contiendra .TRUE. si l'unité logique est connectée à un fichier, .FALSE. sinon.

num : variable entière qui contiendra le numéro d'unité logique auquel est éventuellement connecté un fichier, et n'importe quoi sinon.

nam : variable chaîne de caractères qui contiendra le nom du fichier si le fichier possède un nom, et n'importe quoi sinon.

nmd : variable logique qui contiendra .TRUE. si le fichier possède un nom, .FALSE. sinon.

acc : variable chaîne de caractères qui contiendra 'SEQUENTIAL' ou 'DIRECT' si le fichier est ouvert, et n'importe quoi sinon.

seq : variable chaîne de caractères qui contiendra 'YES' si l'on peut accéder au fichier séquentiellement, 'NO' si c'est impossible et 'UNKNOWN' si cela ne peut être déterminé.

dir : variable chaîne de caractères qui contiendra 'YES' si l'on peut accéder au fichier par accès direct, 'NO' si c'est impossible et 'UNKNOWN' si cela ne peut être déterminé.

frm : variable chaîne de caractères qui contiendra 'FORMATTED' ou 'UNFORMATTED' si le fichier est ouvert, et n'importe quoi sinon.

fmt : variable chaîne de caractères qui contiendra 'YES' si l'on peut accéder au fichier avec format, 'NO' si c'est impossible et 'UNKNOWN' si cela ne peut être déterminé.

unf : variable chaîne de caractères qui contiendra 'YES' si l'on peut accéder au fichier sans format, 'NO' si c'est impossible et 'UNKNOWN' si cela ne peut être déterminé.

blk : variable chaîne de caractères qui contiendra 'ZERO' ou 'NULL' si le fichier est ouvert et possède l'attribut 'FORMATTED', n'importe quoi sinon.

rcl : variable entière qui contiendra la longueur d'un enregistrement (en nombre d'octets) si le fichier est ouvert et est de type 'DIRECT', n'importe quoi sinon.

nxr : variable entière qui contiendra le numéro de l'enregistrement suivant à lire ou à écrire si le fichier est ouvert et est de type 'DIRECT', n'importe quoi sinon.

remarques :

Quel que soit le type d'interrogation, EXIST=exs et OPENED=opn peuvent toujours être renseignés.

Avec une interrogation INQUIRE par nom, les paramètres NAMED=nmd, NAME=nam, SEQUENTIAL=seq, DIRECT=dir, FORMATTED=frm et UNFORMATTED=unf ne sont définis que si le fichier existe. De même, les paramètres NUMBER=num, ACCES=acc, FORM=frm, BLANK=blk, RECL=rcl et NEXTREC=nxr ne sont définis que si le fichier existe et est ouvert.

Avec une interrogation INQUIRE par unité, les paramètres NAMED=nmd, NAME=nam, NUMBER=num, ACCES=acc, SEQUENTIAL=seq, DIRECT=dir, FORM=frm, FORMATTED=frm, UNFORMATTED=unf, BLANK=blk, RECL=rcl et NEXTREC=nxr ne sont définis que si l'unité logique existe et est connectée à un fichier.

Tous les paramètres d'interrogation non répertoriés ci-dessus sont interdits.

1-2.5.9 *Spécifications de format*

Toutes ces spécifications de format peuvent s'utiliser aussi bien dans des instructions FORMAT que directement dans les formats des ordres d'entrée/sortie sous forme de chaîne de caractères constantes ou variables.

Dans ce qui suit, les éléments d, k, n, r, s et w *doivent être des constantes entières*.

autorisées

Iw	pour des entiers
Iw.d	pour des entiers (avec obligation d'afficher au moins d chiffres)
Fw.d	pour des réels sous forme virgule fixe (simple précision)
Ew.d	pour des réels sous forme virgule flottante (simple précision)
Dw.d	pour des réels sous forme virgule flottante (double précision)
Lw	pour des logiques
Aw	pour des chaînes de caractères
A	pour des chaînes de caractères (uniquement à l'édition)
'...'	pour des commentaires (constantes chaînes de caractères)
nH...	pour des commentaires (constantes chaînes de caractères)
X	pour laisser un blanc (ou sauter un caractère en lecture)
sX	pour laisser plusieurs blancs (ou sauter des caractères en lecture)
/	passage au début de la ligne suivante (i.e. fermeture du buffer de ligne)
s(...)	facteur de répétition
kP	facteur d'échelle pour les réels (se combine uniquement avec les spécifications de format F, E ou D)
BN	considérer, dans ce qui suit, les blancs comme des caractères nuls, pour les spécifications I, F, E et D (en lecture uniquement, ignoré en écriture)
BZ	considérer, dans ce qui suit, les blancs comme des caractères zéros, pour les spécifications I, F, E et D (en lecture uniquement, ignoré en écriture)
S	dans la suite du format, le signe + ne sera pas affiché, pour les spécifications I, F, E et D (en écriture uniquement, ignoré en lecture)
SP	dans la suite du format, le signe + sera sytématiquement affiché, pour les spécifications I, F, E et D (en écriture uniquement, ignoré en lecture)
SS	(identique à la spécification S)
Tr	tabulation (positionnement en colonne r); toutefois, *utilisez plutôt sX*
,	séparateur de spécifications (obligatoire comme séparateur dans les listes de spécifications)
:	séparateur de spécifications pouvant se substituer au caractère virgule (peut marquer la fin d'exploration d'un format)

interdites

Ew.dEe	pour les réels (simple précision)
Dw.dEe	pour les réels (double précision)
Qw.d	pour les réels (quadruple précision)
Gw.d	pour les réels (format général)
Gw.dEe	pour les réels (format général)
Ow	pour les entiers et les caractères (sous forme octale)
Zw	pour les entiers et les caractères (sous forme hexadécimale)
"..."	pour les commentaires
TLr	tabulation gauche
TRr	tabulation droite
\	pour maintenir une position sur la ligne courante (en mode interactif)

Ne pas utiliser d'instruction FORMAT vide : par exemple, 100 FORMAT ( ).

remarques

Il faut *impérativement* respecter le même type de données entre spécifications de format et données manipulées en entrée/sortie (par exemple, ne pas éditer une variable réelle avec une spécification de format Iw).

A l'intérieur d'un format, toutes les spécifications de format doivent *obligatoirement* être séparées par des virgules (par exemple, 5XI6 est interdit, il faut coder 5X,I6). Une exception, si plusieurs caractères / se suivent, il n'est pas nécessaire de suivre cette règle (par exemple, /// est permis).

La différence entre virgule (,) et deux points (:) comme séparateur de spécifications de format est la suivante : quand la liste des variables est épuisée avant la liste des spécifications de format, FORTRAN arrête l'exploration du format dès qu'il rencontre un caractère deux points (:), alors qu'il continue de prendre en compte tous les éléments de pagination qui restent si les séparateurs sont des caractères virgule (,).

autorisés

formats variables :

L'utilisation des formats variables rangés dans des zones chaînes de caractères (variables ou tableaux) est tout à fait possible et peut largement être exploitée. Il est également très facile de les construire dynamiquement avec les outils de manipulations de chaînes de caractères disponibles.

autorisé

format de sortie destinée à l'impression :

Un format d'impression ne doit pas dépasser 132 caractères par ligne imprimée (133 avec le caractère de "contrôle du saut du papier"). Il ne faut surtout pas croire qu'une ligne trop longue sera automatiquement coupée en deux au moment de son impression : le plus souvent cela provoque une erreur durant l'exécution du programme, qui s'arrête aussitôt.

L'utilisation de caractères de contrôle du saut du papier est obligatoire dans tout format d'instruction de sortie destinée à l'impression : *chaque ligne effectivement imprimée doit comporter, comme premier caractère, un caractère de contrôle du papier* conforme à ceux figurant ci-dessous.

En particulier, si des sauts de lignes (/) apparaissent au milieu d'un format, il ne faut pas oublier d'introduire un nouveau caractère de saut du papier au début de chaque nouvelle ligne imprimée. Par exemple :

```
(1H1,5X,I3,///,1H0,F10.4)
  |                |
caractères de contrôle du saut du papier
```

La manière d'introduire ces caractères importe peu : '...', nH..., sX, etc... Ce qui important, c'est qu'ils soient présents.

caractères autorisés pour le contrôle du saut du papier :

(blanc) passage au début de la ligne suivante

0 laisser une ligne blanche avant de passer au début de la ligne suivante

1 passage au début de la page suivante

caractères interdits pour le contrôle du saut du papier :

\+ surimpression

\- laisser deux lignes blanches avant de passer au début de la ligne suivante

remarque

Dans le monde micro-informatique, il existe pratiquement toujours, soit une option de compilation, soit un paramètre de l'instruction OPEN, permettant de préciser si l'on utilise ou non de tels caractères de pagination. Pour des questions de compatibilité avec les autres systèmes, il faut impérativement mentionner que l'on utilise ces caractères de contrôle du saut du papier, même si dans OPEN cela conduit à coder un paramètre non autorisé (vérifier auparavant qu'il n'existe pas une option de compilation ayant le même effet).

1-2.6 *INSTRUCTIONS DIVERSES*

autorisée : INCLUDE (*hors norme 77*)

seule forme admise : INCLUDE *'nom-fichier'*

interdites :
%INCLUDE
%GLOBAL
%OPTION

%*instruction* (pour toute instruction)

$*instruction* (pour toute instruction)

OPTION
OPTIONS
COMPILER
EJECT
EDIT
DELETE

DEBUG
END DEBUG
TRACE ON
TRACE OFF
AT
DISPLAY

ENCODE
DECODE

Plus généralement, est interdite toute instruction ne figurant pas parmi les instructions autorisées.

remarque L'instruction INCLUDE, bien que non normalisée, peut être intéressante pour le développement et la diffusion de programmes. Veillez bien à n'employer que la forme admise par MODULAD.
Si l'on préfère s'en tenir à une portabilité quasi-certaine, il est *préférable de ne pas recourir à cette instruction INCLUDE*.

1-3 *PROCEDURES ET BIBLIOTHEQUES "STANDARD" FORTRAN-77*

Il s'agit de répertorier ici les procédures SUBROUTINE ou FUNCTION qui peuvent être utilisées dans le cadre de MODULAD et de mentionner, dans la mesure du possible, celles à exclure.

1-3.1 *FONCTIONS DE LA BIBLIOTHEQUE "STANDARD"*

autorisées :

FONCTION	DEFINITION DE LA FONCTION	TYPE	
		argument	résultat
ABS (x)	valeur absolue	générique	générique
MOD (x,y)	fonction modulo	générique	générique
SIGN((x)	signe	générique	générique
DIM (x,y)	différence positive	générique	générique
MAX (x,y,...)	maximum	générique	générique
MIN (x,y,...)	minimum	générique	générique
INT (x)	conversion en INTEGER	générique	int
AINT (x)	troncature (partie entière)	générique	real/dble
REAL (x)	conversion en REAL	générique	real/dble
DBLE (x)	conversion en DOUBLE PRECISION	générique	dble
DPROD (x,y)	produit en double précision de 2 réels simple précision	real real	dble
NINT (x)	arrondi (résultat sous forme d'entier)	générique	int
ANINT (x)	arrondi (résultat sous forme de réel)	générique	real/dble
EXP (x)	exponentielle	générique	générique
LOG (x)	logarithme (base e)	générique	générique
LOG10 (x)	logarithme (base 10)	générique	générique
SQRT (x)	racine carrée	générique	générique
COS (x)	cosinus	générique	générique
SIN (x)	sinus	générique	générique
TAN (x)	tangente	générique	générique
ACOS (x)	arc cosinus	générique	générique
ASIN (x)	arc sinus	générique	générique
ATAN (x)	arc tangente	générique	générique
ATAN2 (x,y)	arc tangente détermination principale	générique	générique
COSH (x)	cosinus hyperbolique	générique	générique
SINH (x)	sinus hyperbolique	générique	générique
TANH (x)	tangente hyperbolique	générique	générique
CMPLX (x,y)	formation d'un nombre complexe à partir de deux réels	générique	complexe
CONJG (x)	conjugué d'un nombre complexe	complexe	complexe
IMAG (x)	partie imaginaire	complexe	réel
CHAR (x)	obtention d'un caractère à partir du code ASCII (EBCDIC)	char	int
ICHAR (x)	position d'un caractère dans la séquence ASCII (EBCDIC)	int	char
INDEX (x,y)	position d'une chaine y dans une chaine x	char char	int
LEN (x)	longueur d'une chaine de caractères	char	int
LGE (x,y)	comparaison de deux chaines x >= y	char char	logic
LGT (x,y)	comparaison de deux chaines x > y	char char	logic
LLE (x,y)	comparaison de deux chaines x <= y	char char	logic
LLT (x,y)	comparaison de deux chaines x < y	char char	logic

interdites : Est interdite toute fonction "standard" ne figurant pas dans la liste des fonctions autorisées (voir tableau ci-dessus). En particulier, les fonctions de manipulation de bits sont interdites.

interdites : (suite) Sont, entre autres, interdites les fonctions suivantes :

fonctions INTRINSIC à remplacer par des noms de fonctions génériques :

AIMAG (x)	(voir la fonction générique IMAG)
ALOG (x)	(utiliser la fonction générique LOG)
ALOG10 (x)	(utiliser la fonction générique LOG10)
AMAX0 (x,...)	(utiliser la fonction générique MAX)
AMAX1 (x,...)	(utiliser la fonction générique MAX)
AMIN0 (x,...)	(utiliser la fonction générique MIN)
AMIN1 (x,...)	(utiliser la fonction générique MIN)
AMOD (x,y)	(utiliser la fonction générique MOD)
CABS (x)	(voir la fonction générique ABS)
CCOS (x)	(voir la fonction générique COS)
CDABS (x)	(voir la fonction générique ABS)
CDCOS (x)	(voir la fonction générique COS)
CDEXP (x)	(voir la fonction générique EXP)
CDLOG (x)	(voir la fonction générique LOG)
CDSIN (x)	(voir la fonction générique SIN)
CDSQRT (x)	(voir la fonction générique SQRT)
CEXP (x)	(voir la fonction générique EXP)
CLOG (x)	(voir la fonction générique LOG)
CSIN (x)	(voir la fonction générique SIN)
CSQRT (x)	(voir la fonction générique SQRT)
DABS (x)	(utiliser la fonction générique ABS)
DACOS (x)	(utiliser la fonction générique ACOS)
DASIN (x)	(utiliser la fonction générique ASIN)
DATAN (x)	(utiliser la fonction générique ATAN)
DATAN2 (x,y)	(utiliser la fonction générique ATAN2
DCMPLX (x,y)	(voir la fonction générique CMPLX)
DCONJG (x)	(voir la fonction générique CONJG)
DCOS (x)	(utiliser la fonction générique LOG)
DCOSH (x)	(utiliser la fonction générique COSH)
DCOTAN (x)	(voir la fonction générique COTAN)
DDIM (x,y)	(utiliser la fonction générique DIM)
DEXP (x)	(utiliser la fonction générique EXP)
DFLOAT (x)	(utiliser la fonction générique REAL)
DIMAG (x)	(voir la fonction générique IMAG)
DINT (x)	(utiliser la fonction générique INT)
DLOG (x)	(utiliser la fonction générique LOG)
DLOG10 (x)	(utiliser la fonction générique LOG10)
DMAX1 (x,...)	(utiliser la fonction générique MAX)
DMIN1 (x,...)	(utiliser la fonction générique MIN)
DMOD (x,y)	(utiliser la fonction générique MOD)
DNINT (x)	(utiliser la fonction générique NINT)
DREAL (x)	(utiliser la fonction générique REAL)
DSIGN (x)	(utiliser la fonction générique SIGN)
DSIN (x)	(utiliser la fonction générique SIN)
DSINH (x)	(utiliser la fonction générique SINH)
DSQRT (x)	(utiliser la fonction générique SQRT)
DTAN (x)	(utiliser la fonction générique TAN)
DTANH (x)	(utiliser la fonction générique TANH)
I2ABS (x)	(utiliser la fonction générique ABS)
I2DIM (x,y)	(utiliser la fonction générique DIM)
I2MAX0 (x,...)	(utiliser la fonction générique MAX)
I2MIN0 (x,...)	(utiliser la fonction générique MIN)
I2MOD (x,y)	(utiliser la fonction générique MOD)
I2NINT (x)	(utiliser la fonction générique NINT)
I2SIGN (x)	(utiliser la fonction générique SIGN)
IABS (x)	(utiliser la fonction générique ABS)
IDIM (x,y)	(utiliser la fonction générique DIM)
IDINT (x)	(utiliser la fonction générique INT)
IDNINT (x)	(utiliser la fonction générique NINT)
IFIX (x)	(utiliser la fonction générique INT)
ISIGN (x)	(utiliser la fonction générique SIGN)
MAX0 (x,...)	(utiliser la fonction générique MAX)
MAX1 (x,...)	(utiliser la fonction générique MAX)
MIN0 (x,...)	(utiliser la fonction générique MIN)
MIN1 (x,...)	(utiliser la fonction générique MIN)
SNGL (x)	(utiliser la fonction générique REAL)

Chaque fois qu'un nom de fontion 'générique' existe pour une fonction donnée, il est *obligatoire d'utiliser ce nom générique*

interdites :
(suite)

manipulations bit à bit :

IAND (x,y)
IEOR (x,y)
IOR (x,y)
NOT (x)

BTEST (x,y)
IBCLR (x,y)
IBSET (x,y)
ISHFT (x,y)

IBCHNG (x,y)

ISHA (x,y)
ISHC (x,y)
ISHL (x,y)

fonctions numériques diverses :

COTAN (x)
HFIX (x)
INT1 (x)
INT2 (x)
INT4 (x)
INTC (x)
JFIX (x)
NEAREST (x,y)

générateurs de nombres aléatoires :

RND ()
RRAND ()
RANDS (x)

fonction liée à des allocations dynamiques de mémoire :

ALLOCATED (x)

fonction de test de fin de fichier :

EOF (x) (utiliser le paramètre END= de l'instruction READ)

manipulation de chaînes de caractères :

CHARNB (x)
INDEX (x,y,z) (*seule la forme* INDEX (x,y) *est autorisée*)
LEN_TRIM (x)
NBLANK (x)
SCAN (x,y[,z])
VERIFY (x)

fonctions liées à des manipulations d'adresses mémoire :

HUGE (x)
LOC (x)
LOCFAR (x)
LOCNEAR (x)
OFFSET (x)
POINTER (x,y)
SEGMENT (x)

interdites :
(suite)

obtention de certaines valeurs numériques "limites" :

EPSILON (x)
MAXEXPONENT (x)
MINEXPONENT (x)
PRECISION (x)
TINY (x)

1-3.2 *PROCEDURES LIEES AUX BIBLIOTHEQUES "STANDARD"*

autorisée : aucune

interdites : Tous les sous-programmes de type subroutine faisant partie d'une bibliothèque "standard" liée à un compilateur FORTRAN-77 sont interdites. Aucun ne fait partie de la norme 77.

En particulier, les procédures suivantes (s'y trouvent également quelques fonctions) :

gestion d'horloge, de la date et de l'heure :

CLOCK
CLOCKX
CPUTIME
DATE
DATIM
DATIMX
GETTIM
SETTIM
GETDAT
SETDAT
TIME
TIMER
XUFLOW

gestion des anomalies de calcul :

DVCHK
INVALOP
OVEFL
OVERFL
UNDFL
UNDERO

arguments passés à un programme principal :

GETARG
GETCL
NARGS

fin de programme et code retour :

EXIT
SYSRCS
SYSRCT
SYSRCX

entrée/sortie :

FILEINF
FLUSH
IGETER
ICLRER
UNTANY
UNTOFD

interdites :
(suite)

générateurs de nombres aléatoires :

RANDOM
SEED

environnements multitâches :

BEGINTHREAD
DSPTCH
ENDTHREAD
NTASKS
SHRCOM
SYNCRO
THREADID

gestion des erreurs et outils de mise au point :

CDUMP
CPDUMP
DUMP
ERRMON
ERROR
ERRSAV
ERRSET
ERRSTR
ERRTRA
PDUMP
SDUMP
SYSABD
SYSABN

procédures diverses :

ASSIGNM
ENSSTK
INTRUP
PROMPT
PRECFILL
SYSTEM
_USERNIT
_USERTRM

procédures graphiques :

Toutes les procédures de bibliothèques graphiques sont exclues des normes MODULAD.

2-ème partie

REGLES COMPLEMENTAIRES CONCERNANT L'UTILISATION DU FORTRAN-77

Dans la partie précédente, sont détaillés les types de données, les fonctions et les instructions définis comme portables au sens des normes MODULAD, mais il est nécessaire de préciser certaines règles d'utilisation de ces éléments.

2-1 *AFFECTATIONS ET INITIALISATIONS*

2-1.1 *AFFECTATION DE DONNEES NUMERIQUES, LOGIQUES OU CHAINES DE CARACTERES*

Lors d'une affectation, il est préférable d'avoir un type de données identique entre la variable recevant l'affectation et l'expression qui lui est affectée. Néanmoins cette règle n'est pas impérative car, en FORTRAN-77, certaines conversions sont réalisées automatiquement (voir le tableau ci-dessous résumant les conversions lors des affectations).

Toutefois, l'affectation I=X pour X=3.0 peut éventuellement donner la valeur 2 à I, de même I=INT(X). On se protège d'une telle anomalie en écrivant I=X+EPS ou I=INT(X+EPS), où EPS est ici un nombre strictement positif plus petit que 1 (0.1 ou 0.5). Le mieux est encore d'utiliser I=NINT(X).

		type de l'expression à droite du signe d'affectation				
		INTEGER	REAL	DBLE PRECISION	LOGICAL	CHARACTER
type de la variable	INTEGER	affectation	conversion (INT)	conversion (INT)	-	-
	REAL	conversion (REAL)	affectation	conversion (REAL)	-	-
	DBLE PRECISION	conversion (DBLE)	conversion (DBLE)	affectation	-	-
	LOGICAL	-	-	-	affectation	-
	CHARACTER	-	-	-	-	affectation

Tableau des conversions lors d'une affectation <var>=<exp>

2-1.2 *AFFECTATION D'UNE ETIQUETTE A UNE VARIABLE*

Seule l'instruction ASSIGN <constante-étiquette> TO <variable-entière> peut être utilisée pour affecter une constante étiquette à une variable entière (ne jamais utiliser l'affectation simple <var>=<étiquette>). De plus, une telle variable ne doit servir que pour recevoir des constantes étiquettes liées à des instructions FORMAT, permettant alors d'utiliser des formats différents dans une même instruction d'entrée/sortie.

exemple :

```
      ASSIGN 1010 TO NETIQ
        .
        .
        .
      ASSIGN 1020 TO NETIQ
        .
        .
        .
      WRITE (IFICH,NETIQ) ...
        .
        .
        .
1010  FORMAT (...)
1020  FORMAT (...)
```

Par contre, l'instruction GOTO assigné, qui est liée à l'instruction ASSIGN, est interdite.

2-1.3 *INITIALISATIONS PAR DEFAUT*

Il n'existe aucune initialisation par défaut en FORTRAN-77. Quand il est nécessaire d'initialiser une variable avec une valeur précise, il faut le faire à l'aide d'une instruction exécutable d'affectation <var>=<valeur initiale> .

2-1.4 *INITIALISATIONS LORS DES DECLARATIONS*

interdit : Il est interdit d'initialiser une variable lors de sa déclaration à l'aide d'instructions de la forme :

```
INTEGER  SOMME/0/, N/25/
REAL  EPS/1E-8/, PI/3.1416/, T(10)/10*1.0/
```

remarque : Pour respecter une programmation claire, seules des variables dont les valeurs ne changeraient jamais en cours de traitement seraient susceptibles de recevoir une telle initialisation. Mais alors, autant utiliser le principe des constantes nommées définies dans des instructions PARAMETER qui sont justement faites pour ça.

exemple :

```
PARAMETER  (PI=3.141592, EPS=1E-8)
```

En effet, si une variable dont le contenu évolue doit recevoir une initialisation, cette initialisation doit être réalisée au début de la séquence de traitement où elle intervient et à l'aide d'une instruction d'affectation exécutable.
Cette remarque vaut aussi, bien évidemment, pour l'instruction DATA qui de ce fait est également à proscrire.

2-1.5 *INITIALISATIONS A L'AIDE DE DATA*

interdit : L'instruction DATA est interdite pour plusieurs raisons.

La première recoupe directement la partie *remarque* du paragraphe précédent.

La seconde est dûe au fait qu'une initialisation DATA dans un sous-programme n'est pas réalisée chaque fois que l'on active ce sous-programme, mais simplement une fois au début du traitement de l'ensemble d'un programme. Ceci peut être une grave source d'anomalies et manque singulièrement de clarté. C'est également valable pour les initialisations accompagnant des déclarations, qui sont aussi à rejeter (voir paragraphe précédent).

remarque : Les initialisations lors des déclarations ainsi qu'à l'aide de DATA sont donc totalement proscrites. Il suffit de les remplacer par des affectations dans les instructions exécutables. C'est plus clair, ce n'est pas plus long à écrire et c'est pratiquement aussi rapide en exécution (en tout cas cela ne mettra sûrement pas en péril les performances d'une application).

2-1.6 *ENTITE DE PROGRAMME BLOCK DATA*

interdit : Si l'instruction DATA est interdite, il est évident que l'instruction BLOCK DATA, et l'entité de programme qui lui est attachée, sont interdites. En effet, le rôle de BLOCK DATA est de permettre des initialisations de type DATA pour des variables devant figurer dans des COMMON (rappel d'une règle FORTRAN-77 : il est interdit de voir figurer dans des instructions DATA des variables présentes dans des instructions COMMON, ailleurs que dans une entité d'initialisation BLOCK DATA; et en dehors des instructions BLOCK DATA et END, une telle entité ne doit renfermer que des instructions DATA, PARAMETER, SAVE, IMPLICIT, DIMENSION, <type>, COMMON ou EQUIVALENCE).

remarque : L'initialisation des variables placées dans des instructions COMMON, quand elle est nécessaire, doit être réalisée à l'aide d'instructions d'affectation en début de programme principal.

2-2 *EXPRESSIONS*

2-2.1 *OPERATEURS*

autorisés :

**	puissance
*	multiplication
/	division
+	addition
-	soustraction
//	concaténation de chaines de catactères
.GT.	supérieur à (utiliser LGT pour les chaines de caractères)
.GE.	supérieur ou égal à (utiliser LGE pour les chaines de caractères)
.LT.	inférieur à (utiliser LLT pour les chaines de caractères)
.LE.	inférieur ou égal à (utiliser LLE pour les chaines de caractères)
.EQ.	égal à
.NE.	différent de

autorisés : (suite)

.NOT.	négation logique
.AND.	et logique
.OR.	ou logique inclusif
.EQV.	équivalence logique
.NEQV.	ou logique exclusif

interdits :

```
>
>=
<
<=
.XOR.
```

2-2.2 *PRIORITE DES OPERATEURS*

OPERATEURS	PRIORITE
appel de fonction	1
**	2
* /	3
+ -	4
.GE. .GT. .LE. .LT. .EQ. .NE.	6
.NOT.	7
.AND.	8
.OR.	9
.EQV. .NEQV.	10

Priorité des opérateurs (contexte numérique)
(1 est le plus haut niveau de priorité)

OPERATEURS	PRIORITE
appel de fonction	1
//	5
.GE. .GT. .LE. .LT. .EQ. .NE.	6
.NOT.	7
.AND.	8
.OR.	9
.EQV. .NEQV.	10

Priorité des opérateurs (contexte caractères)
(1 est le plus haut niveau de priorité)

2-2.3 *REGLES POUR L'EVALUATION DES EXPRESSIONS*

FORTRAN évalue en premier les expressions placées sur le plus haut niveau de parenthèses (i.e. les plus imbriquées).

Lorsque deux opérateurs sont placés sur le même niveau de parenthèses, FORTRAN effectue en premier l'opération de plus haut niveau de priorité, et si les deux opérateurs sont de même niveau, alors l'expression est évaluée de gauche à droite.

Pour que la dernière règle puisse s'appliquer quel que soit le compilateur, il est *interdit* d'écrire A**B**C ; il faut toujours écrire A**(B**C) ou (A**B)**C.

Parfois certains compilateurs se permettent des libertés là où les règles de priorité ne jouent plus vraiment, en particulier vis-à-vis d'un ordre d'évaluation des expressions imposé par la présence de parenthèses (ceci le plus souvent pour des questions d'optimisation).

exemples : L'expression (X+0.5)+0.5 peut être optimisée en X+1.0 .
L'expression C+FONC(A) , où C est en COMMON et se trouve modifié par l'appel à la fonction FONC, peut donner des résultats différents selon les compilateurs.

solution : Les expressions dont la valeur peut varier suivant l'ordre des opérations, doivent être calculées en plusieurs étapes en utilisant des variables intermédiaires.

remarque : Contrairement à une opinion assez répandue, le fait de décomposer des calculs en s'aidant de variables intermédiaires ne favorise pas spécialement les optimisations. Pour ces questions d'optimisation, il est nettement préférable de laisser agir les compilateurs seuls plutôt que tenter de les aider.

En revanche, recourir à des variables intermédiaires peut éventuellement augmenter la lisibilité d'un calcul.

2-2.4 *EXPRESSIONS MIXTES*

Pour les expressions dites "mixtes" qui mélangent des opérateurs arithmétiques, des opérateurs sur chaînes de caractères, des opérateurs de comparaison et/ou des opérateurs logiques, il faut améliorer la lisibilité des expressions et supprimer les ambiguïtés de priorité en ajoutant des parenthèses.

exemples :

```
IF (X.GT.Y-1) THEN
```

doit plutôt s'écrire

```
IF (X.GT.(X-1)) THEN
```

```
TEST=A.LT.B.AND.X.GT.5
```

doit plutôt s'écrire

```
TEST=(A.LT.B).AND.(X.GT.5)
```

2-2.5 *EXPRESSIONS CHAINES DE CARACTERES*

La longueur des chaînes de caractères et des expression manipulées ne doit jamais dépasser 255 caractères.
Seules les constantes chaînes de caractères définies entre caractères apostrophes '...' sont utilisables. Les constantes de type Hollerith nH.... peuvent éventuellement apparaître, mais uniquement dans des formats.

interdit : On ne doit jamais trouver une variable ou un même élément de tableau de type chaînes de caractères à la fois à droite et à gauche d'un signe d'affectation.

exemple :

```
CHARACTER*10  MOT
      .
      .
      .
MOT='JOUR'
MOT='BON'//MOT
```

est *interdit*.

En effet, une majorité de compilateurs fournissent des résultats totalement imprévisibles. Il suffit de passer par une zone intermédiaire.

Pour les manipulations de chaînes de caractères, les fonctions CHAR, ICHAR, INDEX et LEN sont disponibles. De même, pour extraire ou manipuler une sous-chaîne de caractères, il suffit d'utiliser la syntaxe :

chaîne([*exp1*]:[*exp2*])

où *chaîne* est une variable ou un élément de tableau de type chaîne de caractères, *exp1* et *exp2* des constantes, des variables ou des expressions entières.

exp1 désigne le caractère de début de la sous-chaîne
exp2 désigne le caractère de fin de la sous-chaîne

exemple :

```
CHARACTER*7  ZONE
      .
      .
      .
ZONE='BONSOIR'
      .
      .
      .
WRITE (IOUT,*)  ZONE(4:5)     donne     SO
WRITE (IOUT,*)  ZONE(3:3)     donne     N
WRITE (IOUT,*)  ZONE(:3)      donne     BON
WRITE (IOUT,*)  ZONE(4:)      donne     SOIR
```

Les manipulations de chaînes de caractères en entrée/sortie avec format se font à l'aide de la spécification de format Aw.

A noter que pour une écriture, le format A tout seul suffit, FORTRAN ajustant la longueur d'écriture à la longueur de la chaîne écrite.

exemple :

```
CHARACTER*5  MOT
      .
      .
      .
MOT='BANAL'
      .
      .
      .
WRITE (IOUT,'(1HO,A)')  MOT     donne     BANAL
WRITE (IOUT,'(1HO,A3)')  MOT    donne     BAN
WRITE (IOUT,'(1HO,A7)')  MOT    donne       BANAL
```

La dernière instruction WRITE ci-dessus écrit BANAL sur 7 caractères en alignant à droite dans la zone, la chaîne de caractères. La précédente tronque à 3 caractères cette même chaîne, sur sa droite.

En revanche, pour une lecture il faut toujours préciser une longueur w derrière une spécification de format Aw.

exemple :

```
CHARACTER*5  MOT          données à lire          HELAS
      .                                           HELAS
      .                                           HELAS
      .
READ (5,'(A5)')  MOT      zone lue      HELAS
      .                                   .
      .                                   .
READ (5,'(A3)')  MOT      zone lue      HEL AS
      .                                   .
      .                                   .
READ (5,'(A7)')  MOT      zone lue      HELAS      valeur récupérée   LAS
      .                                   .
```

2-2.6 *CONVERSIONS LORS DE L'EVALUATION DES EXPRESSIONS (MELANGE DE TYPES)*

Il est bien entendu préférable d'éviter les mélanges de types de données dans les expressions. Cependant, en cas de nécessité, FORTRAN réalise automatiquement certaines conversions. Ces conversions sont résumées dans le tableau suivant qui précise le type du résultat quand l'opération est possible.

		second opérande				
		INTEGER	REAL	DBLE PRECISION	LOGICAL	CHARACTER
premier opérande	INTEGER	INTEGER	REAL	DBLE PRECISION	-	-
	REAL	REAL	REAL	DBLE PRECISION	-	-
	DBLE PRECISION	DBLE PRECISION	DBLE PRECISION	DBLE PRECISION	-	-
	LOGICAL	-	-	-	LOGICAL	-
	CHARACTER	-	-	-	-	CHARACTER

Tableau des conversions lors d'une opération <opérande-1> <opérateur> <opérande-2>
(le type du résultat est indiqué à l'intersection d'une ligne et d'une colonne)

remarque : Recommandations à propos de l'opérateur puissance :

X**N est du type de X si N est entier

A**B est de type réel si A et B sont réels (simple ou double précision)

Les autres cas sont à éviter.

2-2.7 *PRECISION DANS LES EXPRESSIONS LORS DE CALCULS NUMERIQUES*

* limites pour un entier :

Un entier est compris dans l'intervalle : <entier> ∈ [-2147483648, +2147483647]
(-2**31) (2**31-1)

Pour les machines 16-bits, déclarez les variables en INTEGER*4, s'il n'existe aucune autre possibilité pour obtenir des INTEGER de longueur 4 (le plus souvent, il existe une option de compilation permettant de choisir, avec ces types d'architectures de machines, entre des INTEGER de longueur 2 et des INTEGER de longueur 4).

* limites pour un réel :

Un réel, simple ou double précision, sera limité aux intervalles suivants :

<réel> ∈ [-1E+35, -1E-35] U {0.} U [+1E-35, +1E+35]

Ces valeurs restrictives garantissent une portabilité sur toutes les machines.

Dans la pratique, on définira deux valeurs RELMIN=1E-35 et RELMAX=1E+35 , qui serviront de référence pour les comparaisons.

2-2.8 *COMPARAISONS NUMERIQUES*

Pour les comparaisons de deux réels avec un test d'égalité, certaines machines peuvent donner des résultats inattendus. Il est donc recommandé de tester indirectement l'égalité de deux réels :

exemple :

```
IF (A.EQ.B) ...
```

se codera plutôt

```
IF (ABS(A-B).LT.EPS) ...
```

où EPS est un nombre positif très petit, pouvant dépendre de la précision de la de la comparaison ou de la machine (par exemple, EPS=1E-8 ou bien EPS=1E-35).

2-2.9 *COMPARAISONS LOGIQUES*

Il est *interdit* d'utiliser des données de type logique dans des expressions de comparaison. Les données logiques ne peuvent apparaître que dans des expressions logiques (opérateurs logiques).

exemple :

```
LOGICAL  TEST1, TEST2, RESUL
   .
   .
RESUL=TEST1.GT.TEST2
```

est *interdit*.

2-2.10 *COMPARAISONS DE CHAINES DE CARACTERES*

Il faut impérativement éviter d'utiliser les opérateurs de comparaison .GT., .GE., .LT. et .LE. avec des chaines de caractères et utiliser à la place les fonctions proposées à cet effet : LGT, LGE, LLT et LLE. Ces fonctions réalisent les comparaisons en utilisant toujours le code de représentation interne ASCII et ceci sur tout type de machine.

En revanche, les opérateurs de comparaison .EQ. et .NE. sont parfaitement utilisables car ils ne sont pas tributaires de la représentation interne des caractères.

exemple :

```
CHARACTER*5  MOT1, MOT2
   .
   .
MOT1='BRAVO'
MOT2='MERCI'
   .
   .
IF (LLE(MOT1,MOT2))  ...
```

2-3 TABLEAUX ET VARIABLES INDICEES

Le nombre d'indice que l'on peut utiliser pour des tableaux est limité à *cinq*. Toute variable indicée doit être référencée avec autant d'indices qu'il en a été déclaré dans l'instruction de dimensionnement.

2-3.1 *DIMENSIONNEMENT DES TABLEAUX*

2-3.1.1 *Comment réaliser le dimensionnement des tableaux*

autorisé : Dimensionnez les tableaux avec des instructions de déclaration explicite de type :

```
INTEGER
REAL
DOUBLE PRECISION
LOGICAL
CHARACTER
```

interdit : Il est interdit de dimensionner des tableaux à l'aide des instructions suivantes :

```
DIMENSION
COMMON
```

remarque : L'ordre DIMENSION n'est pas autorisé : il faut utiliser des types explicites. Ceci est lié au fait que tous les objets doivent être désormais déclarés.

Pour une raison identique de clarté et de lisibilité des programmes, les tableaux utilisés dans des COMMON doivent être dimensionnés hors des COMMON, donc en recourant également à des types explicites.

2-3.1.2 *Dimensionnement de tableaux dans les sous-programmes*

Les tableaux dont la taille est fixe (et pas trop importante) peuvent être déclarés localement dans les procédures SUBROUTINE ou FUNCTION. Sinon, ils sont placés comme arguments/paramètres ou bien, à la rigueur, passés par COMMON.

2-3.1.3 *Dimensionnement ajustables des paramètres dans les sous-programmes*

obligatoire : Si un tableau est passé comme argument, ses dimensions doivent aussi être passées en argument.

S'il s'agit d'un tableau passé comme paramètre avec des dimensions ajustables, il faut toujours utiliser X(N) et jamais X(1), ni X(*).

interdit : Dans un sous-programme, un tableau passé en argument/paramètre ne doit jamais avoir ses dimensions transmises à l'aide d'un COMMON.

Pour les dimensionnements ajustables, il est possible de remplacer le dernier indice d'un tableau par une étoile (*). Cette technique est *interdite* pour les types de tableaux, dans le cadre des développements MODULAD. En revanche, elle est *autorisée* pour l'ajustement de la longueur d'une chaine de caractères passée en argument/paramètre sous le forme CHARACTER ZONE*(*) .

2-3.1.4 *Allocation dynamique de mémoire pour des tableaux*

FORTRAN-77 n'admet pas l'allocation dynamique de mémoire. Cependant, il est possible de simuler un pseudo-système d'allocation dynamique de tableaux dans des "super-tableaux" en s'appuyant sur des passages arguments/paramètres. Avec une telle technique d'allocation dynamique de tableaux dans des super-tableaux, il est impératif de respecter scrupuleusement le type des variables lors des passages arguments/paramètres (par exemple, pas de redéfinition d'un tableau d'entiers sur des réels). Il faut donc prévoir autant de "super-tableaux" de taille fixe, dans le programme principal, qu'il y a de types de données (INTEGER, REAL, DOUBLE PRECISION, LOGICAL et CHARACTER) et réaliser les allocations de tableaux, par recouvrement mémoire dans les sous-programmes, en respectant ces types de données.

Un tel système de gestion dynamique de la mémoire doit reposer sur un ensemble de sous-programmes spécifiques (dans le cadre des développements MODULAD, une telle bibliothèque existe).

2-3.2 *TYPE ET FORME DES INDICES*

autorisés : Les indices ayant la forme suivante :

- constante entière
- variable entière (indicée ou)
- expression entière (sans restriction)

interdit : Tout indice qui n'est pas de type entier.

remarque : Les indices de tableaux peuvent ne pas démarrer de 1 : en particulier, ils peuvent être négatifs.

exemple :

```
INTEGER  TAB(-2:2)
REAL  ALPHA(0:10)
      .
      .
      .
I=-1
TAB(I)=406
      .
ALPHA(0)=-14.27
      .
```

attention : En aucun cas, un indice d'un tableau ne doit sortir du champ de variation défini, pour cet indice, lors du dimensionnement.

2-3.3 *VARIABLES INDICEES ET SOUS-CHAINES DE CARACTERES*

Pour les tableaux de chaînes de caractères, les indices des tableaux doivent toujours figurer avant la désignation des sous-chaînes de caractères.

exemple :

```
CHARACTER*10  ZONE(20)
      .
      .
      .
ZONE(15)='BONJOUR'
      .
      .
      .
ZONE(15)(4:7)='SOIR'
```

Pour le dimensionnement des tableaux de chaînes de caractères, il est possible de permuter indifféremment les attributs de longueur de chaîne et de taille de tableau.

exemple :

```
CHARACTER  NOM*20(10)
CHARACTER  PRENOM(10)*20
```

C'est également vrai si l'on mélange, dans des sous-programmes, tableaux de dimension ajustable avec chaînes de caractères de longueur ajustable.

exemple :

```
CHARACTER  ZONE1*(*)(N)
CHARACTER  ZONE2(N)*(*)
```

2-3.4 *RANGEMENT DE MATRICES DANS DES TABLEAUX MONODIMENSIONNES*

Ces formes de rangement n'ont pas un caractère obligatoire, d'autant qu'elles vont à l'encontre des normes précédentes de MODULAD. Cependant, il est souhaitable de suivre ces nouvelles règles car elles rejoignent la forme de rangement des tableaux utilisée dans tous les autres langages de programmation.

Soit Aij le terme général de la matrice, avec i indice des lignes et j indice des colonnes.

* matrice pleine :

A11 A12 A13 ... A1n (1ère ligne) A21 A22 A23 ... A2n (2ème ligne) ...

* matrice symétrique :

A11 (1ère ligne) A21 A22 (2ème ligne) A31 A32 A33 (3ème ligne) ... An1 An2 An3 ... Ann (n-ième ligne)

* matrice diagonale :

A11 A22 A33 ... Ann

* matrice scalaire :

A (avec Aij=0 si i non égal à j, et Aij=A si i égal à j)

2-4 INSTRUCTIONS AVEC BRANCHEMENTS

2-4.1 *INSTRUCTIONS GOTO*

2-4.1.1 *Instruction GOTO inconditionnel*

interdite : L'instruction GOTO est *interdite*, sauf pour reconstituer les modèles de programmation structurée DOWHILE...ENDDO, REPEAT...UNTIL et BREAK qui manquent en FORTRAN-77 (se reporter à la partie traitant des modèles de programmation structurée).

2-4.1.2 *Instructions GOTO assigné et GOTO calculé*

interdites : Les instructions GOTO assigné et GOTO calculé sont *interdites* car ce sont des instructions à branchements multiples qui vont totalement à l'encontre de la programmation structurée. Il faut les ramener à d'autres figures de programmation autorisées.

exemples :

```
      ASSIGN  230  TO  NETIQ
      GOTO  NETIQ
230   C=138
      INDICE=2
      GOTO  (330,310,350,390)  INDICE+1
350   D=161
```

sont *interdits*.

2-4.2 *INSTRUCTIONS IF*

2-4.2.1 *Instruction IF arithmétique*

interdite : L'instruction IF arithmétique est *interdite* car elle conduit à des branchements multiples.

2-4.2.2 *Instruction IF logique*

interdite : L'instruction IF logique, sous la forme IF (...) GOTO est *interdite*, sauf pour reconstituer les modèles de programmation structurée DOWHILE...ENDDO et REPEAT...UNTIL qui manquent en FORTRAN-77 (se reporter à la partie traitant des modèles de programmation structurée).

remarque : Une instruction IF logique, de la forme IF (...) <instruction> , est autorisée à condition que <instruction> ne soit pas une instruction GOTO ou une instruction IF arithmétique.

L'instruction IF (...) THEN est pleinement autorisée.

2-5 BOUCLES DO

2-5.1 *BOUCLE DO AVEC INDEX*

syntaxe : La syntaxe générale de la boucle DO avec index est la suivante :

```
      DO  n  <index>=<borne-1>,<borne-2>[,<pas>]
        .
        .
        .
   n    CONTINUE
```

L'<index> de la boucle doit être une variable simple (i.e. une variable non indicée) de type entier ou réel. Les trois paramètres <borne-1>, <borne-2> et <pas> peuvent être des constantes, des variables ou des expressions de type entier ou réel. En particulier, le <pas> peut être positif ou négatif, entier ou fractionnaire (dans ce cas, attention au type de l'<index> et au test d'arrêt avec les problèmes d'arrondis).

Par défaut, et quelles que soient les valeurs de <borne-1> et <borne-2>, la valeur du <pas> est +1.

Avec un <pas> positif, si <borne-1> est strictement plus grand que <borne-2> ou bien, avec un <pas> négatif, si <borne-1> est strictement plus petit que <borne-2> alors FORTRAN-77 ne pénètre pas dans le corps de la boucle et passe directement à l'exécution de la première instruction qui suit le corps de la boucle.

exemple :

```
        .
      NDEB=11
      NFIN=10
        .
        .
      Y=117
      DO  30  I=NDEB,NFIN
        .
        .
        .
   30 CONTINUE
      X=406
        .
```

dans ce cas, après l'instruction Y=117, c'est directement l'instruction X=406 qui est exécutée, sans que l'on soit passé dans la boucle.

obligatoire : A chaque boucle DO doit obligatoirement être associée une instruction CONTINUE.

exemples :

```
        .
      DO  20  I=1,N
        DO  10  J=1,N
        .
        .
        .
   10   CONTINUE
   20 CONTINUE
        .
```

mais les deux exemples suivants

```
      DO  30  I=1,N
      DO  30  J=1,N
          .
          .
30    CONTINUE
```

et

```
      DO  40  I=1,N
      DO  40  J=1,N
          .
          .
40    A(I,J)=2*X
```

sont *interdits*.

obligatoire : Si vous voulez disposer de la valeur de l'<index> d'une boucle DO en dehors de cette boucle (qu'elle se soit terminée normalement ou que vous soyez sorti avant la fin), il faut absolument passer par une variable auxiliaire à laquelle vous affecterez la valeur de cet <index> à l'intérieur de la boucle. Cette précaution est nécessaire car certains compilateurs, pour des questions d'optimisation, ne conservent pas les variables index de boucles en mémoire, mais uniquement dans des registres internes de calcul.

interdits : Il ne faut pas modifier la valeur de l'<index> d'une boucle DO à l'intérieur de cette boucle.

Il est interdit de pénétrer à l'intérieur d'une boucle sans être passé par l'instruction DO. En particulier, il est interdit de sortir d'une boucle DO sans l'avoir terminée, pour effectuer une séquence de calcul à l'extérieur et revenir aussitôt dans cette boucle. De toute façon l'instruction GOTO est interdite (voir les règles régissant l'utilisation de l'instruction GOTO).

exemples :

```
      GOTO 50
          .
          .
      DO  90  I=1,N
          .
          .
50      X=X+2
          .
          .
90    CONTINUE
```

ou encore

```
      DO  120  I=1,K
          .
          .
        GOTO  150
          .
          .
110     X=X+2
          .
          .
120   CONTINUE
          .
          .
          .
150   Y=98
      Z=117
      A=138
      GOTO  110
```

sont *interdits*.

2-5.2 *BOUCLE DO AVEC INDEX ET SANS ETIQUETTE*

interdite : La boucle DO avec index, utilisée sans étiquette, est interdite.

exemple :

```
      DO I=1,N
       .
       .
      END DO
```

est *interdit*.

2-5.3 *BOUCLE DO WHILE*

interdite : La boucle DO WHILE, bien qu'assez répandue en FORTRAN aujourd'hui, ne fait pas pas partie de la norme 77. Cette forme de boucle est donc exclue des normes MODULAD.

exemple :

```
      DO  WHILE (A.GT.0)
       .
       .
      END DO
```

est *interdit*.

2-6 *OPERATIONS D'ENTREE/SORTIE*

Dans la mesure du possible, il faut essayer de ne pas effectuer d'opérations d'entrée/sortie dans des modules de calcul numérique. Ces opérations doivent être réalisées dans des sous-programmes spécifiques, en distinguant quand c'est possible :

- lecture des paramètres
- lecture des données
- lecture/écriture des résultats intermédiaires
- écriture des résultats

2-6.1 *NUMEROS D'UNITES LOGIQUES*

Les numéros d'unités logiques doivent obligatoirement être rangés dans des variables entières pour pouvoir être utilisés dans les instructions d'entrée/sortie (les constantes sont interdites).

Par souci de clarté et de portabilité, ces variables contenant les numéros d'unités logiques à utiliser, devront toutes être initialisées en début de programme principal et transmises aux sous-programmes soit à l'aide d'instructions COMMON, soit sous forme de passages arguments/paramètres.

Les numéros d'unités logiques 1 à 9 doivent être réservés pour les entrées/sorties 'standard' du type consoles, imprimantes, etc.... Pour les entrées/sorties sur supports magnétiques, on ne doit utiliser que les numéros de 10 à 99.

Il est interdit d'ouvrir un fichier uniquement à l'aide de son numéro d'unité logique, c'est-à-dire sans préciser l'option FILE= avec un nom, dans l'instruction OPEN.

2-6.2 *OUVERTURE ET FERMETURE DES FICHIERS*

Toute exploitation d'un fichier, en lecture ou en écriture, doit obligatoirement être précédée d'un OPEN avant sa première utilisation afin de l'ouvrir, et se terminer par un CLOSE pour le fermer. De plus, il faut toujours veiller à ouvrir un fichier le plus tard possible, par rapport à la logique de la programmation, et le fermer le plus tôt possible.

obligatoire : Dans une instruction OPEN doivent toujours figurer le paramètre UNIT=, mais surtout le paramètre FILE=. Il est interdit de manipuler un fichier qui n'aurait pas de nom. Les autres paramètres sont à coder en fonction des contraintes imposées par le traitement.

remarque : Le paramètre STATUS est très délicat à manipuler car il peut ne pas être pris en compte. En effet, avec certains systèmes d'exploitation, coder la paramètre STATUS= n'a aucun effet et peut être source de problèmes. C'est le cas de STATUS='NEW', 'SCRATCH' ou 'OLD' avec OPEN, et de STATUS='DELETE' ou 'KEEP' avec CLOSE. Pour un OPEN, seul STATUS='UNKNOWN' peut être utilisé sans problème. Avec CLOSE, dans la plupart des cas, STATUS='KEEP' peut être utilisé.

2-6.3 *FICHIERS A ACCES SEQUENTIELS*

Pour l'exploitation des fichiers séquentiels, les seuls ordres de lecture et d'écriture à utiliser sont READ et WRITE. Le repositionnement au début d'un fichier séquentiel se fait à l'aide de REWIND.

obligatoire : Entre une ouverture (OPEN) et fermeture (CLOSE), un fichier séquentiel ne peut être exploité qu'en lecture ou bien en écriture, mais pas les deux simultanément.

Il faut impérativement faire suivre tout OPEN d'un fichier séquentiel d'un REWIND car l'instruction OPEN ne repositionne pas obligatoirement un fichier à son début.

Il faut respecter le type de manipulation avec format ou sans format, que vous avez éventuellement défini dans l'instruction OPEN à l'aide du paramètre FORM='FORMATTED' ou 'UNFORMATTED', et ne pas mélanger des ordres READ et WRITE avec et sans format sur le même fichier.

Pour les fichiers sans format, les ordres d'entrée/sortie READ et WRITE doivent toujours comporter le même nombre et le même type d'éléments dans les listes d'éléments à lire ou à écrire qui se correspondent.

recommandé : Dans l'instruction OPEN, il est préférable de toujours faire figurer comme paramètres, en plus de UNIT= et FILE=, les mots-clés ACCESS= (avec 'SEQUENTIAL' pour un séquentiel) et FORM= (avec 'FORMATTED' ou 'UNFORMATTED').

exemples :

```
OPEN (UNIT=IFICH,FILE=NOMFIC,ACCESS='SEQUENTIAL',FORM='FORTATTED')
```

ou bien

```
OPEN (UNIT=IFICH,FILE=NOMFIC,ACCESS='SEQUENTIAL',FORM='UNFORTATTED')
```

2-6.4 *FICHIERS A ACCES DIRECTS*

Pour l'exploitation de fichiers à accès direct, contrairement aux fichiers séquentiels, il est parfaitement possible, entre un OPEN et un CLOSE, à la fois de lire et d'écrire.

obligatoire : Dans l'instruction OPEN, en plus de UNIT= et FILE=, les paramètres ACCESS= (avec 'DIRECT') et RECL= (avec RECL=<longueur enregistrement> en nombre d'octets) sont à coder obligatoirement. Comme pour les fichiers séquentiels, il est recommandé de toujours préciser FORM= (avec 'FORMATTED' ou 'UNFORMATTED').

exemples :

```
OPEN (UNIT=IFICH,FILE=NOMFIC,ACCESS='DIRECT',FORM='FORTATTED',RECL=LENREG)
```

ou bien

```
OPEN (UNIT=IFICH,FILE=NOMFIC,ACCESS='DIRECT',FORM='UNFORTATTED',RECL=LENREG)
```

En lecture, comme en écriture, il faut toujours préciser explicitement le numéro de l'enregistrement que l'on souhaite manipuler (REC=<numéro> dans READ et WRITE). Attention, il n'y a aucune incrémentation automatique de ce numéro. Le premier enregistrement est numéroté 1.

exemples :

```
READ (UNIT=IFICH,....,RECL=NUMENR) ...
```

ou bien

```
WRITE (UNIT=IFICH,....,RECL=NUMENR) ...
```

interdit : En dehors des instructions OPEN/CLOSE et READ/WRITE, aucune autre instruction ne doit être utilisée avec les fichiers à accès direct (pas de REWIND possible).

2-6.5 *TESTS DE FIN DE FICHIERS*

Disposer d'une fonction logique pour détecter les fins de fichiers n'est pas normalisé en FORTRAN-77. Le seul test admis est de s'aider du paramètre END= dans l'instruction READ pour les fichiers séquentiels (pas de test de fin de fichier en accès direct).

obligatoire : Pour tester la fin d'un fichier séquentiel, il faut opérer de la façon suivante en s'inspirant de la reconstitution d'une boucle DOWHILE...ENDDO de programmation structurée telle qu'elle est proposée dans la suite de ce document.

exemple :

```
      .
140   CONTINUE
        READ (UNIT=IFICH,....,END=190) ...
          .
          .
          .
        GOTO  140
190   CONTINUE
      .
```

interdit : Tout recours à une fonction logique de test de fin de fichier est interdit.

2-6.6 *GESTION DES ERREURS D'ENTREE/SORTIE*

La détection des erreurs d'entrée/sortie en FORTRAN-77 n'est normalisée que sémantiquement. Par conséquent, un même programme peut réagir différemment face à ce type d'anomalies en fonction du système d'exploitation sur lequel il s'exécute : en particulier, les paramètres ERR= et IOSTAT= n'ont pas toujours un fonctionnement parfaitement coordonné. Cependant nous pouvons proposer une approche pragmatique de cette question pour les développements MODULAD.

recommandé : Pour les instructions OPEN, CLOSE, INQUIRE et éventuellement REWIND, il est souhaitable de coder systématiquement les paramètres ERR= et IOSTAT= en s'inspirant de la démarche suivante, qui ne résoud cependant pas le problème de l'activation de ERR= sans pour autant fournir à travers IOSTAT= une valeur autre que 0.

exemples :

```
      OPEN (UNIT=IFICH,FILE=NOMFIC,...,ERR=120,IOSTAT=IOS)
120   IF (IOS.NE.0)  CALL ERREUR (....)
      .
      .
      CLOSE (UNIT=IFICH,ERR=230,IOSTAT=IOS)
230   IF (IOS.NE.0)  CALL ERREUR (....)
      .
```

Dans le cadre des développements MODULAD, tester systématiquement les erreurs d'entrée/sortie pour les instructions READ et WRITE, ne s'impose pas. Toutefois, si dans un programme de tels tests étaient réalisés, il y aurait lieu de les coder en s'inspirant du modèle ci-dessus.

2-7 *SOUS-PROGRAMMES*

Seul les sous-programmes de type SUBROUTINE ou FUNCTION sont admis. Les fonctions définies sous forme d'une seule instruction, au début d'une entité de programme, ne doivent pas être utilisées.

2-7.1 *STRUCTURE DES SOUS-PROGRAMMES*

Les normes MODULAD imposent de respecter les points suivants pour l'écriture des sous-programmes, avec comme conseil supplémentaire qu'un sous-programme ne doit pas être trop long (environ 4 à 5 pages maximum) : ceci devrait être possible sans pour autant remettre en cause le principe d'un découpage modulaire à visée fonctionnnelle.

obligatoire : Le nom d'un sous-programme, SUBROUTINE ou FUNCTION, ne doit pas dépasser 6 caractères et ne doit comporter que de lettres ou des chiffres, le premier caractère étant obligatoirement alphabétique. De plus, ce nom ne doit pas correspondre à un mot-clé FORTRAN, ni à aucun nom de sous-programme d'une quelconque bibliothèque "standard" liée à un compilateur (une liste assez complète est fournie avec la liste des fonctions autorisées).

Les sous-programmes ne peuvent être récursifs.

Les variables "locales" à un sous-programme doivent être réinitialisées à chaque entrée dans le sous-programme (il ne faut surtout supposer que les valeurs sont conservées d'un appel à l'autre, sauf dans le cas où les variables possèdent l'attribut SAVE).

2-7.2 *PASSAGES ARGUMENTS/PARAMETRES*

2-7.2.1 *Terminologie*

Nous distinguons les notions d'argument et de paramètre.

* Les arguments sont les éléments que l'on trouve au point d'appel d'un sous-programme (dans l'instruction CALL pour l'appel d'une SUBROUTINE, ou lors de la référence à une fonction pour l'appel d'une FUNCTION).

* Les paramètres sont les éléments que l'on trouve au point d'entrée d'un sous-programme (dans les instructions SUBROUTINE ou FUNCTION).

exemples :

```
CALL  CALCUL (A,B,C)                        A,B,C sont des arguments

GAMMA=2*SOMME(ALPHA,BETA)+1                 ALPHA,BETA sont des arguments

SUBROUTINE  CALCUL (X,Y,Z)                  X,Y,Z sont des paramètres
END

REAL FUNCTION  SOMME (PARM1,PARM2)          PARM1,PARM2 sont des paramètres
END
```

2-7.2.2 *Règles à respecter pour les passages arguments/paramètres*

obligatoire : Le nombre d'arguments doit toujours être identique au nombre de paramètres.

Arguments et paramètres qui se correspondent doivent être de même type.

Les arguments sont obligatoirement des variables ou des noms de sous-programmes déclarés EXTERNAL ou INTRINSIC (pas de constante ou d'expression dans un point d'appel à un sous-programme).

A priori, les dimensionnements des tableaux qui se correspondent comme arguments/paramètres doivent être identiques (exception faite pour les techniques d'allocation dynamique de mémoire de tableaux dans un super-tableaux mises en oeuvre à l'aide d'appels de sous-programmes).

Il ne faut pas utiliser de notation * pour l'ajustement d'une dimension d'un tableau reçu sous forme de paramètre. Pour les tableaux de taille ajustable placés comme paramètres, le dimensionnement ajustable doit obligatoirement être réalisé à l'aide de variables entières, elles-mêmes passées comme arguments/paramètres. La notation * peut être utilisée, mais uniquement pour l'ajustement de la longueur de paramètres chaînes de caractères de longueur ajustable.

exemples :

```
SUBROUTINE  CALCUL (N,P,...,A,B,C,...)
INTEGER  N, P
REAL  A (20,10), B(N,P), Z(N)
```

et

```
SUBROUTINE  TEXTE (...,MOT,...)
CHARACTER  MOT*(*)
```

sont *autorisés*.

```
SUBROUTINE  CALCUL (N,...,X,Y,Z,...)
REAL  X (20,*), Y(N,*), Z(*)
```

est *interdit*.

recommandé : Les arguments et les paramètres doivent être disposés de la façon suivante dans les listes d'appels et de points d'entrée :

arguments d'entrée	(=paramètres en entrée non modifiables)
arguments d'entrée et de sortie	(=paramètres en entrée modifiables)
arguments de sortie	(=paramètres fournis uniquement en sortie)
arguments de travail	(=paramètres pour des stockages temporaires)
arguments de contrôle/validité	(=paramètres pour la gestion des anomalies)

et chaque groupe d'arguments et de paramètres doit être séparé du suivant par un blanc ou un passage à la ligne.

Pour échanger des données, il faut toujours privilégier, chaque fois que cela est possible, un passage sous forme argument/paramètre plutôt que par COMMON.

interdit : Il ne faut pas faire de mélanges de types de données entre arguments et paramètres.

L'utilisation de constantes, de constantes PARAMETER ou d'expressions comme arguments est interdite.

La notation * pour le dimensionnement ajustable de paramètres de type tableau est interdite (seule la longueur d'un paramètre chaîne de caractères peut être codée *).

Il est interdit d'utiliser de points de retour définis à l'aide d'étiquettes dans la liste des arguments d'un sous-programme SUBROUTINE. Le seul point de retour possible pour un sous-programme SUBROUTINE est la première instruction qui suit son point d'appel. Il existe une exception à cette règle dans le cas de la gestion des erreurs à l'aide d'un sous-programme spécifique (se reporter au conseils pour le développement de programmes dans la suite de ce document).

2-7.3 *ATTRIBUTS EXTERNAL ET INTRINSIC*

L'attribut EXTERNAL possède deux significations différentes.

2-7.3.1 *Attribut EXTERNAL pour un passage argument/paramètre*

Appliqué à un nom de sous-programme défini par le programmeur (SUBROUTINE ou FUNCTION), il indique que ce nom de sous-programme sera passé comme argument dans un appel de sous-programme.

obligatoire : Si vous souhaitez passer comme argument un nom de sous-programme, il faut, dans le programme appelant, le déclarer EXTERNAL s'il s'agit d'un nom de sous-programme utilisateur.

exemple :

```
EXTERNAL  CALCUL
CALL  APPEL (...,CALCUL,...)
SUBROUTINE  APPEL (...,SSP,...)
CALL  SSP (...)          ------->  (réalise un appel à CALCUL)
END
SUBROUTINE  CALCUL (...)  <-------
END
```

2-7.3.2 *Attribut EXTERNAL pour une référence externe*

Si l'attribut EXTERNAL porte sur un nom de sous-programme faisant partie de la bibliothèque standard liée à un compilateur, ceci signifie que, dans cette entité de programme, vous faites référence à un sous-programme "externe", c'est-à-dire un sous-programme de même nom, mais redéfini de façon explicite.

obligatoire : Si vous souhaitez définir un sous-programme utilisateur portant le nom d'un sous-programme SUBROUTINE ou FUNCTION (générique ou intrinsèque) présent dans la bibliothèque "standard" d'un compilateur, vous devez le déclarer explicitement à l'aide de l'attribut EXTERNAL dans l'entité de programme concernée. La portée de cet attribut EXTERNAL est limitée à la portée de ce nom de sous-programme : ainsi il est parfaitement possible, dans une autre entité de programme de continuer à faire appel, par ce même nom, au sous-programme standard.

exemple :

```
      .
      EXTERNAL  ABS
      .
      S=2*ABS(A,B)                    ------>  (appel à la fonction ABS
      .                                        définie programmeur)
      CALL  SUITE (....)
      .
      CALL  ENCORE (....,ABS,....)
      .
      .
      .
      REAL  FUNCTION  ABS (X,Y)       <------
      REAL  X, Y
      ABS=X+Y
      END
      .
      .
      .
      SUBROUTINE  SUITE (....)
      .
      ALPHA=ABS(BETA)                 ------>  (appel à la fonction
      .                                        "standard" ABS,
      END                                      valeur absolue)
      .
      .
      .
      SUBROUTINE  ENCORE (....,FONC,....)
      .
      C=FONC(D,E)                     ------>  (appel à la fonction ABS
      .                                        définie programmeur)
      END
```

2-7.3.3 *Attribut INTRINSIC pour un passage argument/paramètre*

L'attribut INTRINSIC permet d'indiquer qu'un nom de fonction intrinsèque ou générique d'une bibliothèque "standard" sera passé comme argument lors d'un appel de sous-programme.

obligatoire : Si vous souhaitez passer comme argument un nom de fonction intrinsèque ou générique figurant dans une bibliothèque "standard", vous devez, dans le programme appelant, le déclarer INTRINSIC.

attention : Si vous déclarez INTRINSIC un nom de fonction "standard" générique, vous lui faites perdre sa propriété générique.

interdit : Un nom de fonction standard ne peut posséder en même temps les attributs EXTERNAL et INTRINSIC.

Il existe des restrictions sur les noms de fonctions "standard" pouvant recevoir l'attribut INTRINSIC. L'attribut INTRINSIC est interdit pour les noms de fonctions "standard" suivants :

INT	IDINT	HFIX	IFIX			
MAX	MAXO	AMAXO	MAX1	AMAX1	DMAX1	
MIN	MINO	AMINO	MIN1	AMIN1	DMIN1	
FLOAT	REAL	DREAL	SNGL	DBLE	CMPLX	DCMPLX
CHAR	ICHAR					
LGE	LGT	LLE	LLT			
NOT	IAND	IOR	IEOR			
ISHFT	BTEST	IBSET	IBCLR			

De toute façon, parmi ces fonctions, seules sont utilisables, dans le cadre de MODULAD, les fonctions INT, MAX, MIN, REAL, DBLE, CHAR et ICHAR. La restriction ci-dessus porte donc surtout sur ces fonctions.

exemple :

```
      INTRINSIC  ABS
      .
      CALL  APPEL (...,ABS,...)
      .
      SUBROUTINE  APPEL (...,FONC,...)
      .
      X=FONC(Y)                    ------->  (appel à la fonction "standard"
      .                                       ABS, valeur absolue)
      END
```

2-7.4 *INSTRUCTION SAVE*

Il est possible de conserver d'un appel sur l'autre les valeurs d'une variable "locale" à un sous-programme, à condition de la déclarer SAVE. A priori toutes les variables FORTRAN sont statiques ce qui dispense théoriquement d'utiliser l'instruction SAVE. Toutefois certains compilateurs optimisent la place mémoire occupée par un programme en récupérant la place mémoire des variables "locales" aux sous-programmes non actifs.

Par définition, un paramètre ou une variable COMMON ne peuvent figurer dans une instruction SAVE.

Il existe plusieurs types de sauvegarde : peuvent être déclarés SAVE aussi bien une variable, un tableau, que l'ensemble d'un COMMON local nommé; l'instruction SAVE permet également de sauvegarder l'ensemble de tout les zones mémoire locales à un sous-programme. Nous précisons ci-dessous ce qui est admis pour MODULAD.

obligatoire : Si vous voulez conserver, d'un appel à l'autre, la valeur d'une variable locale à un sous-programme, vous devez impérativement l'indiquer à l'aide d'une instruction SAVE.

exemple :

```
      SUBROUTINE  SOMMER (...)
      REAL  X, Y(12)
      INTEGER  N
      .
      SAVE  X,Y,N
      .
```

interdit : L'instruction SAVE "blanc" (sauvegarde de tout ce qui est local à un sous-programme) est interdite.

L'instruction SAVE portant sur un COMMON nommé local à un sous-programme est également interdite. La justification de ce type de SAVE ne vaut pas pour MODULAD puisqu'il est interdit de faire apparaître des COMMON locaux à des sous-programmes sans les avoir également définis dans le programme principal (cette précaution assure toujours de fait leur sauvegarde).

Aucun paramètre ni aucune variable placée dans un COMMON ne doit figurer dans un SAVE.

exemples :

```
      SUBROUTINE  CALCUL (....)
      SAVE
```

et

```
      SUBROUTINE  APPEL (....)
      COMMON  /BLOC1/ A,B,C
      SAVE  /BLOC1/
```

sont *interdits*.

2-8 MISE EN COMMUN DE DONNEES

La mise en commun de données entre sous-programmes peut se faire par le biais de passages arguments/paramètres ou bien à travers le partage de zones mémoire défini à l'aide d'instructions COMMON.

2-8.1 *PRINCIPES D'UTILISATION DE L'INSTRUCTION COMMON*

Il faut limiter le plus possible l'utilisation d'instructions COMMON qui peuvent être source d'erreurs multiples et poser des problèmes de maintenance, et toujours leur préférer des passages arguments/paramètres.

Quelques types de données sont plus particulièrement candidats à être mis en commun à travers des instructions COMMON, par exemple des numéros d'unités logiques liés à des fichiers.

exemple :

```
      INTEGER  IN, OUT
      COMMON  /INOUT/ IN, OUT
      .
      READ (IN,....)  ...
      WRITE (OUT,....)  ...
```

Quelques autres exemples figurent dans la suite de ce document, sous forme de conseils de programmation.

Contrairement à un idée largement répandue, il est totalement faux de croire que l'instruction COMMON permet le dimensionnement variable ou ajustable de tableaux.

2-8.2 *REGLES D'UTILISATION DE L'INSTRUCTION COMMON*

Pour l'utilisation de COMMON dans le cadre de MODULAD, il faut respecter les règles suivantes.

obligatoire : Seuls sont admis les instructions COMMON avec un nom.

Le nom d'un COMMON nommé est limité à 6 caractères maximum et ne doit comporter que des lettres ou des chiffres, le premier caractère étant obligatoirement alphabétique.

Il faut placer les variables chaînes de caractères dans des COMMON spécifiques où ne doivent figurer aucun autre type de variables.

Il faut ne faire figurer qu'un seul bloc nommé par instruction COMMON.

Dans un même bloc nommé mis en en commun par COMMON, on doit toujours avoir la même liste de variables, avec rigoureusement les mêmes noms d'identificateurs, le même nombre de variables, les mêmes types et les mêmes dimensions.

Les tableaux doivent être dimensionnés en dehors des instructions COMMON à l'aide de types explicites.

Tout COMMON nommé présent dans un sous-programme doit obligatoirement apparaître comme tel dans le programme principal de cette application. En effet, avec certains compilateurs ou certains systèmes d'exploitation, l'intégrité d'une zone COMMON n'est garantie que si elle est présente dans le programme principal.

exemples :

```
      REAL  PI, VALMAX
      CHARACTER  TEXTE*100

      COMMON  /VAL/ PI, VALMAX
      COMMON  /TEXT/ TEXTE
```

recommandé : Pour des problèmes d'alignement mémoire, il est déconseillé de mélanger des variables de types différents dans un même COMMON. Si néanmoins un mélange se fait, il faut impérativement ranger les variables dans l'ordre suivant : DOUBLE PRECISION, REAL, INTEGER et LOGICAL.

exemple :

```
      REAL  X, Y, Z
      INTEGER  IN, OUT

      COMMON  /XYZ/ X, Y, Z
      COMMON  /INTOU/  IN, OUT
```

est *recommandé*, plutôt que

```
      REAL  X, Y, Z
      INTEGER  IN, OUT

      COMMON  /ZONE/ X, Y, Z, IN, OUT
```

interdit : L'utilisation d'instruction COMMON "blanc" est interdite.

L'instruction DATA étant interdite, l'initialisation à l'aide de DATA (dans une entité BLOCK DATA) de variables placées dans des COMMON est forcément interdite, et de fait l'entité BLOCK DATA...END également. Il suffit de recourir à des instructions d'affectation, en créant au besoin un sous-programme spécifique.

Il est interdit de faire figurer dans un même COMMON des variables de type caractères avec des variables d'un autre type.

Il est interdit de calquer, sur une même zone COMMON, des variables de types différents d'un sous-programme à l'autre.

Il est interdit de modifier le dimensionnement de tableaux passés en COMMON d'un sous-programme à l'autre.

Il est interdit de faire figurer explicitement les attributs de dimension d'un tableau dans une instruction COMMON. Les tableaux mis en commun doivent être dimensionnés à l'aide d'instructions fixant un type explicite DOUBLE PRECICION, REAL, INTEGER, LOGICAL ou CHARACTER.

L'utilisation, dans un sous-programme, de tout bloc COMMON nommé ne figurant pas dans le programme principal est interdit.

exemples :

```
REAL  PI, VALMAX
CHARACTER  TEXTE*100

COMMON  /BLOC1/ A, B,  /BLOC2/ X, Y
COMMON  /BLOC3/ N, T(25)
COMMON  /BLOC4/ TEXTE, PI, VALMAX
COMMON  ALPHA, BETA, GAMMA
```

sont *interdits*.

2-9 INCLUSION DE FICHIERS

L'inclusion de fichiers "source" peut être envisageable, pour les développements MODULAD, bien que cette instruction ne fasse pas partie de la norme FORTRAN-77. Il subsiste néanmoins un inconvénient majeur à son emploi du fait que l'on trouve plusieurs syntaxes, parfois fort différentes. Toutefois une syntaxe parmi d'autres semble s'imposer.

Cette possibilité, restreinte à la seule phase de développement d'un programme, facilitera le maintien de la cohérence interne d'une application au regard des instructions COMMON qu'elle contient. On pourra inclure par INCLUDE tous les COMMON nommés ainsi que les déclarations qui doivent les accompagner.

autorisé : La seule syntaxe admise, et uniquement pour des programmes en cours de développement ou de mise au point est la suivante :

```
INCLUDE  <nom fichier>
```

où *<nom fichier>* peut prendre diverses formes selon les systèmes.

Pour l'inclusion de COMMON nommés, il peut être pratique d'éclater en deux fichiers séparés les COMMON nommés et les déclarations des variables mises en commun.

interdit : L'utilisation de l'instruction INCLUDE, dans le cadre MODULAD, doit être impérativement restreinte au développement et à la mise au point de programmes. Dans un programme diffusé par MODULAD, il ne doit plus subsister aucune instruction INCLUDE : la structure des noms de fichier *<nom fichier>* diffère beaucoup trop d'un système d'exploitation à l'autre pour que l'on puisse envisager la moindre portabilité.

3-ème partie

REGLES ET CONSEILS POUR LE DEVELOPPEMENT ET LA PROGRAMMATION

Il n'est pas question de décrire ici les différentes méthodes de développement, ni de détailler leurs avantages et leurs inconvénients. En outre, la plupart des développeurs scientifiques y sont assez peu sensibles ou accoutumés.

Lorsque l'on développe un logiciel, après avoir défini ce que l'on attend de ce logiciel, c'est-à-dire fixé ses spécifications externes, il faut procéder à l'étude de ses spécifications internes. Etablir des spécifications internes conduit à préciser en détail comment chacune des fonctions attendues du logiciel sera réalisée et comment ses différentes fonctions s'articuleront entre elles. Vient ensuite la phase de rédaction des programmes constituant le logiciel, qui sera suivie d'une phase de test puis de validation.

A ces différents stades de développement d'un logiciel, trois catégories de critères doivent être envisagés et attentivement examinés : critères de qualité, critères de découpage, critères de lisibilité.

* critères de qualité :

  adéquation
  fiabilité
  maintenabilité
  évolutivité
  performance
  convivialité

* critères de découpage :

  fonctionnalité (ou stabilité, qui fait quoi)
  évolutivité (faire mieux, plus tard)
  technique (portabilité, condition d'exploitation, environnement spécifique....)
  algorithmique (comment chacun satisfait son objectif)
  taille d'un module (encombrement, place mémoire, facilité de manipulation....)

* critères de lisibilité :

  auto-documentation du code (choix du nom des identificateurs, commentaires....)
  structuration du code (faciliter sa compréhension)
  éviter les "astuces" de programmation difficilement compréhensibles

Dans ce document sont détaillés les critères de lisibilité pour une écriture en FORTRAN-77. Les critères de qualité ne sont pas abordés et seuls quelques éléments concernant les critères de découpage sont envisagés. Pour obtenir plus de détails sur ces questions, nous renvoyons à des ouvrages spécialisés.

3-1 CRITERES DE DECOUPAGE

Dans le domaine scientifique, les critères de découpage fonctionnel et algorithmique n'ont guère lieu d'être distingués. De même, le critère taille des modules est envisagé sous le seul aspect fonctionnel. Concernant les critères de découpage imposés par des contraintes techniques, seules les contraintes de portabilité sont abordées.

3-1.1 DECOUPAGE MODULAIRE FONCTIONNEL

La phase d'analyse, préalable à l'écriture, doit toujours conduire à la définition d'un découpage modulaire fonctionnel de l'application à développer. Si un tel découpage peut se révéler délicat à réaliser en matière de gestion, il ne présente en général guère de difficultés en matière de développement scientifique. Les grandes fonctions ou les grandes phases de traitement sont le plus souvent faciles à identifier et à isoler, et les modules de programmation (terminologie équivalente : entité de programmation, procédure, sous-programme) recoupent assez exactement ce découpage modulaire fonctionnel qui s'impose donc assez "naturellement" au développeur.

Par ailleurs, il n'est guère raisonnable dans le domaine scientifique d'imposer un découpage d'une application en entités de programmation scrictement limitées à la taille d'une page de listage, voire une "page" d'écran (la contrainte étant que le développeur doit pouvoir consulter en un seul coup d'oeil tout un sous-programme) : il faut conserver à une telle entité sa cohérence fonctionnelle même si elle doit correspondre à 3 ou 4 pages de listage. Au-delà, il faut tout de même envisager un découpage plus fin.

3-1.2 DECOUPAGE MODULAIRE A VISEE DE MAINTENANCE

Au découpage modulaire à visée fonctionnelle, nous souhaitons adjoindre une autre perspective, celle d'un découpage à visée de maintenance. Il s'agit d'essayer de repérer, dans une application, les parties plus ou moins sensibles en matière d'évolution et d'isoler, dans des entités de programmation spécifiques, les portions de code suceptibles d'être modifiées rapidement et fréquemment.

3-1.3 DECOUPAGE MODULAIRE A VISEE ADAPTATIVE

La perspective d'un découpage modulaire à visée de maintenance peut également s'accompagner d'un découpage à visée adaptative qui peut en être très proche : son but est de parvenir à regrouper en modules particuliers les éléments de code que l'on sait les plus sensibles en matière de portabilité de façon à minimiser les éventuelles adaptations pour cause de "portage" sur une machine cible donnée.

Une application dans laquelle se trouve ainsi identifiées et localisées des entités potentiellement modifiables, sera bien plus aisée à maintenir, à adapter et à faire évoluer.

Pour ces problèmes de méthodologie et de critères de développement, aucune règle particulière n'est imposée par MODULAD. Nous laissons au développeur la responsabilité du choix de son approche et de sa démarche.

3-2 CRITERES DE LISIBILITE

S'imposer des critères de lisibilité, c'est adopter un style de programmation avec comme objectif de faciliter le développement d'une application, sa mise au point, sa maintenance, son adaptation, son évolution. Dans cette perspective, il s'agit d'augmenter la lisibilité d'un programme en utilisant des règles de présentation précises et de ne recourir qu'à des schémas connus de programmation structurée.

3-2.1 *PRESENTATION DES INSTRUCTIONS*

Il faut coder les instructions de façon à accroître la lisibilité de toute séquence de programme et refléter au mieux la structure de ce programme.

La recommandation la plus importante concerne le recours systématique à l'indentation afin de traduire visuellement une telle structure.

règle : A l'intérieur d'une boucle, toutes les instructions doivent être indentées de 2 colonnes pour les boucles DO et de 4 colonnes pour les autres types de boucles. Derrière IF...THEN toutes les instructions dépendant d'une clause THEN ou ELSE doivent être indentées de 4 colonnes, les mots-clés ENDIF et ELSE de 2 colonnes.

Toutes les règles sur la disposition générale des instructions définies au paragraphe 1-1.2 doivent évidemment être aussi respectées (usage des colonnes, suite d'une instruction, codage des étiquettes, utilisation du blanc comme séparateur).

3-2.2 *UTILISATION DE L'INSTRUCTION GOTO*

Tout programmeur sait aujourd'hui qu'il faut, si possible, bannir de la programmation toute instruction de branchement inconditionnel. L'instruction GOTO est toujours source de difficultés pour la maintenance des programmes. Si la plupart des langages de programmation évolués permettent de s'en affranchir complètement, ce n'est pas le cas du FORTRAN-77 qui, en effet, ne dispose pas de toutes les instructions "classiques" de programmation structurée. Puisqu'il est impossible d'interdire complètement l'emploi de l'instruction GOTO, du moins son utilisation doit-elle être entièrement banalisée et strictement restreinte à la codification des instructions de programmation structurée manquantes (cf. paragraphe 3-2.3).

Dans cette perspective de banalisation de l'instruction GOTO, MODULAD impose le respect de la règle suivante :

règle : une instruction GOTO ne doit jamais renvoyer sur une instruction autre que CONTINUE. Plus généralement, on ne doit jamais rencontrer d'étiquette préfixant une instruction exécutable autre que CONTINUE.

Les règles de programmation structurée conduisent à interdire toute instruction à branchement multiple. Conséquence, les instructions GOTO assigné et GOTO calculé sont bien évidemment interdites.

3-2.3 *LES MODELES DE PROGRAMMATION STRUCTUREE*

En FORTRAN-77, il est parfaitement possible de disposer de toutes les figures de programmation structurée, soit que les instructions correspondantes existent effectivement, soit que l'on puisse aisément les reconstituer, et à défaut d'interdire GOTO, nous pouvons du moins restreindre son usage. Pour chaque modèle retenu, nous donnons sa traduction en FORTRAN-77 dans une forme à respecter impérativement, y compris pour le détail de l'indentation. Ces modèles de programmation structurée doivent constituer la base de toute description algorithmique à l'aide de pseudo-code.

3-2.3.1 *Modèle du test IF*

Guère de problèmes pour la traduction du test IF puisqu'en FORTRAN-77 nous disposons presque directement de la traduction avec l'instruction IF. Nous distinguons trois traduction possibles : IF...ENDIF, IF...ELSE...ENDIF et IF...ENDIF ne comportant q'une seule instruction.

```
pseudo-code :              traduction :              indentation:

IF test                    IF (test) THEN            0 IF (test) THEN
    .                          .                       |   .
    .                          .                     4 |   .
    .                          .                       |   .
ENDIF                      ENDIF                     2 ENDIF


IF test                    IF (test) THEN            0 IF (test) THEN
    .                          .                       |   .
    .                          .                     4 |   .
  ELSE                       ELSE                    2 ELSE
    .                          .                       |   .
    .                          .                     4 |   .
    .                          .                       |   .
ENDIF                      ENDIF                     2 ENDIF


IF test                    IF (test) instruction
  instruction
ENDIF
```

Il faut noter que le mot-clé ENDIF doit se trouver placé en retrait de 2 colonnes par rapport au mot-clé IF dont il dépend. Ceci se justifie eu égard à des dispositions comparables dans d'autres langages comme le C, le PL/1 ou le PASCAL. Cette disposition permet surtout de distinguer un autre usage du mot-clé ENDIF lorsqu'il intervient dans la traduction du modèle de choix multiple SELECT...ENDSEL.

3-2.3.2 *Modèle du choix multiple SELECT...ENDSEL*

Ce modèle de pseudo-code est également connu sous le nom de SELECT-logique, par opposition au SELECT-par-valeur qui correspond au "CASE" du PASCAL ou au "switch/break" du C, mais qui n'est pas un modèle retenu en matière de figure de programmation structurée. Un SELECT-par-valeur peut toujours se traduire par un SELECT-logique.
FORTRAN-77 comporte les éléments de codification de ce modèle avec le mot-clé ELSEIF.

```
pseudo-code :              traduction :              indentation:

SELECT
  WHEN test-1              IF (test-1) THEN          0 IF (test-1) THEN
    .                          .                       |   .
    .                          .                     4 |   .
    .                          .                       |   .
  WHEN test-2              ELSE IF (test-2) THEN     2 ELSE IF (test-2) THEN
    .                          .                       |   .
    .                          .                     4 |   .
    .                          .                       |   .
  WHEN test-3              ELSE IF (test-3) THEN     2 ELSE IF (test-2) THEN
    .                          .                       |   .
    .                          .                     4 |   .
    .                          .                       |   .
  OTHER                    ELSE                      2 ELSE
    .                          .                       |   .
    .                          .                     4 |   .
    .                          .                       |   .
ENDSEL                     ENDIF                     0 ENDIF
```

Ici le mot-clé ENDIF est parfaitement aligné avec IF, contrairement à un simple test. La clause OTHER, traduite par ELSE, n'est pas obligatoire : elle dépend de la logique de la séquence à coder.

remarque : Bien qu'un SELECT-logique revienne à des tests successifs imbriqués, il ne s'agit pas d'une simple imbrication de tests qui se traduirait de la façon suivante :

```
pseudo-code :              traduction :              indentation:

IF test-1                  IF (test-1) THEN          0 IF (test-1) THEN
   IF test-2                  IF (test-2) THEN          IF (test-2) THEN
      .                          .                   4     .
      .                          .                   8     .
      .                          .                         .
    ELSE                       ELSE                  6   ELSE
      .                          .                         .
      .                          .                   8     .
      .                          .                   6     .
   ENDIF                      ENDIF                       ENDIF
 ELSE                       ELSE                      2 ELSE
   IF test-3                  IF (test-3) THEN          IF (test-3) THEN
      .                          .                   4     .
      .                          .                   8     .
      .                          .                         .
    ELSE                       ELSE                  6   ELSE
      .                          .                         .
      .                          .                   8     .
      .                          .                   6     .
   ENDIF                      ENDIF                       ENDIF
ENDIF                      ENDIF                     2 ENDIF
```

3-2.3.3 *Modèle de boucle DOWHILE...ENDDO*

Il n'existe pas de boucle WHILE en FORTRAN-77 et nous proposons la reconstitution suivante :

```
pseudo-code :              traduction :              indentation:

                        n1 CONTINUE                  0 CONTINUE
DOWHILE test               IF (.NOT.(test)) GOTO n2      IF (.NOT.(test)) GOTO n2
   .                       .                         2     .
   .                       .                         4     .
   .                       .                         2     .
ENDDO                      GOTO n1                       GOTO n1
                        n2 CONTINUE                  0 CONTINUE
```

3-2.3.4 *Modèle de boucle REPEAT...UNTIL*

Il n'existe pas non plus de boucle REPEAT...UNTIL et nous proposons comme reconstitution :

```
pseudo-code :              traduction :              indentation:

REPEAT                  n1 CONTINUE                  0 CONTINUE
   .                       .                         4     .
   .                       .                         2     .
   .                       .                               .
UNTIL test                 IF (.NOT.(test)) GOTO n1      IF (.NOT.(test)) GOTO n1
                        CONTINUE                     0 CONTINUE
```

remarque : Certains compilateurs génèrent un avertissement WARNING face à cette séquence où apparait une instruction CONTINUE sans étiquette préfixe : elle sert uniquement à délimiter la fin de la boucle; ainsi toute boucle se termine par CONTINUE. Il n'y a pas lieu d'attacher de l'importance à ce type de message.

3-2.3.5 *Modèle de boucle FOR...ENDFOR*

Ce type de boucle très puissant peut être utilisé sous deux formes différentes dans une description algorithmique à l'aide de pseudo-code, premièrement sous forme descritpive libre, deuxièmement en empruntant une forme comparable à l'instruction "for" du langage C et assez proche d'une boucle DO avec un index en FORTRAN.

Première forme :

```
FOR <description libre d'un processus itératif>
  .
  .
  .
ENDFOR
```

Cette première forme n'implique pas une traduction unique en FORTRAN-77 et peut correspondre à n'importe laquelle des trois boucles retenues.

Deuxième forme :

```
FOR <expression initialisation> ; <expression test "tant que" > ; <expression modification>
  .
  .
  .
ENDFOR
```

Elle correspond au schéma :

```
<expression initialisation>
DOWHILE <expression test "tant que" >
  .
  .
  .
  <expression modification>
ENDDO
```

avec : <expression initialisation> = initialisation d'un index de boucle
<expression test "tant que"> = test pour la poursuite des itérations au sens de "tant que"
<expression modification> = modification de l'index avant de passer à l'itération suivante

Cette deuxième forme appelle une traduction assez naturelle en FORTRAN à l'aide de la boucle DO :

```
pseudo-code :              traduction :                  indentation:

FOR init ; test ; modif    DO n index=borne1,borne2[,pas]  0 |DO n index=borne1,borne2[,pas]
  .                          .                             |
  .                          .                           2 |  .
  .                          .                             |  .
ENDFOR                 n   CONTINUE                      0 |CONTINUE
```

3-2.3.6 *Modèle de rupture de boucle BREAK*

Interrompre une boucle en cours d'exécution est un modèle qui n'existe pas en FORTRAN-77.

```
pseudo-code :              traduction :                  indentation:

boucle                     boucle                        0 |boucle
  .                          .                             |
  .                          .                           2 |  .
  BREAK                      GOTO n                      ou|  GOTO n
  .                          .                           4 |  .
  .                          .                             |  .
finboucle                  finboucle                     0 |finboucle
                       n   CONTINUE                      0 |CONTINUE
```

remarque : Dans ce cas, on ne peut éviter de recourir à une instruction GOTO, mais elle doit renvoyer uniquement à la fin de la boucle, c'est-à-dire uniquement après la dernière instruction de la boucle que l'on cherche à quitter. Comme un GOTO ne doit jamais renvoyer vers autre chose qu'un CONTINUE, cette première instruction qui suit la boucle devient alors obligatoirement un CONTINUE. A la sortie d'une telle boucle, il est souvent nécessaire d'introduire un test afin de savoir si cette boucle s'est terminée normalement ou bien si elle a été interrompue volontairement.

3-2.3.7 *Modèle de boucle de lecture séquentielle avec test de fin de fichier*

Le modèle de parcours séquentiel d'un fichier avec arrêt sur test de fin de fichier se ramène traditionnellement à une boucle de type DOWHILE...ENDDO.

Cependant, en FORTRAN-77, un test de fin de fichier ne s'effectue pas à l'aide d'une fonction spécifique retournant vrai ou faux, mais directement à travers l'instruction de lecture READ. Ceci implique une transcription particulière si l'on veut respecter les règles de programmation structurée :

pseudo-code :

```
   1-ère lecture dans le fichier
DOWHILE test "non fin de fichier"
    .
    .
    traitement
    .
    .
    lecture suivante dans le fichier
ENDDO
```

traduction :

```
n1  CONTINUE
      READ (...,END=n2,...) ...
        .
        .
        traitement
        .
        .
      GOTO n1
n2  CONTINUE
```

indentation:

```
0 |CONTINUE
  |  READ (...,END=n2,...) ...
2 |  .
4 |  .
4 |  traitement
4 |  .
2 |  .
  |GOTO n1
0 |CONTINUE
```

RESUME DES INSTRUCTIONS FORTRAN-77 UTILISABLES

CLASSIFICATION DES INSTRUCTIONS	NATURE DES INSTRUCTIONS	INSTRUCTIONS AUTORISEES	INSTRUCTIONS INTERDITES
définition d'une entité de programme	programme principal	PROGRAM	
	sous-programmes	SUBROUTINE type FUNCTION	ENTRY FUNCTION
			BLOCK DATA
	instruction		instruction fonction
déclarations et types de données (instructions non exécutables)	données entières	INTEGER	INTEGER*n
	données réelles	REAL DOUBLE PRECISION	REAL*n
	données logiques	LOGICAL	LOGICAL*n
	données caractères	CHARACTER*n	CHARACTER
	données complexes	COMPLEX	COMPLEX*n DOUBLE COMPLEX
	données diverses		BYTE FIXED DOUBLE FIXED FRACTIONNAL DOUBLE FRACTIONNAL
	déclarations pour les sous-programmes	EXTERNAL INTRINSIC	INTERFACE TO
	types de données implicites		IMPLICIT IMPLICIT NONE
	déclaration d'une structure		STRUCTURE END STRUCTURE
	déclaration d'une union		UNION END UNION
	délimitation dans une union		MAP END MAP
	insertion d'une structure		RECORD
spécifications (instructions non exécutables)	mise en commun	COMMON avec nom	COMMON blanc
	gestion mémoire	SAVE	AUTOMATIC
	dimensionnement des tableaux	dimensionnement avec type explicite	DIMENSION
	recouvrement de données		EQUIVALENCE
	listes nommées		NAMELIST

CLASSIFICATION DES INSTRUCTIONS	NATURE DES INSTRUCTIONS	INSTRUCTIONS AUTORISEES	INSTRUCTIONS INTERDITES
initialisations	constantes nommées	PARAMETER	
	initialisations		DATA (et donc BLOCK DATA)
instructions de contrôle (instructions exécutables)	affectation	= ASSIGN ... TO ...	
	débranchements	GOTO inconditionnel	GOTO assigné GOTO calculé

	tests	IF logique
		IF ... THEN
		ELSE
		ELSEIF ... THEN
		ENDIF
			IF arithmétique
			
			WHILE

	boucles	DO CONTINUE	DO WHILE ENDDO EXIT CYCLE
	instruction à choix multiples		SELECT CASE ... CASE ... CASE DEFAULT END SELECT
	allocation dynamique de mémoire		ALLOCATE DEALLOCATE
	sous-programmes	CALL RETURN	CHAIN
	fin/pause d'une entité de programme	STOP END	PAUSE WAIT
entrées/sorties (instructions exécutables)	ouverture/fermeture	OPEN CLOSE	
	interrogation	INQUIRE	
	lecture/écriture	READ WRITE	PRINT PUNCH ACCEPT DISPLAY TYPE READ avec NAMELIST WRITE avec NAMELIST
	positionnement dans un fichier	REWIND	BACKSPACE ENDFILE
	définion d'un format	FORMAT	
	entrées/sorties particulières pour des fichiers à accès direct		DEFINE FILE FIND REWRITE DELETE LOCKING UNLOCK

CLASSIFICATION DES INSTRUCTIONS	NATURE DES INSTRUCTIONS	INSTRUCTIONS AUTORISEES	INSTRUCTIONS INTERDITES
instructions diverses	inclusion de fichiers "source"	INCLUDE	%INCLUDE
	options de compilation et mise en forme d'un listage source		%GLOBAL %OPTIONS %instruction $instruction OPTION OPTIONS COMPILER EJECT EDIT DELETE
	instructions de mise au point		DEBUG END DEBUG AT DISPLAY TRACE ON TRACE OFF
	instructions de conversion		ENCODE DECODE