Manuel de Descriptif Informatique
Fascicule D2.02 : Règles de développement
Document D2.02.01

Règles de programmation FORTRAN

Résumé :

Ce document présente les règles retenues par l'Equipe de Développement d'*Aster* (EDA) pour l'écriture des routines du code en FORTRAN 77.

On peut distinguer deux sortes de règles :

- les règles issues du livre "FORTRAN 77 Guide pour l'écriture de programmes portables" (F. FICHEUX-VAPNE et Coll). Pour ces règles, nous avons parfois modifié l'énoncé en transformant un conseil en règle,
- les règles résultant de l'expérience acquise pendant les premières années du projet. Ces dernières règles sont parfois dictées par des choix techniques stratégiques : par exemple, la gestion de la mémoire (JEVEUX) dispense souvent de l'usage du COMMON. Ces dernières règles ont donc une portée moins générale que les premières.

Le respect de ces règles a deux objectifs principaux :

- assurer une bonne portabilité du code,
- assurer une bonne lisibilité (et donc maintenabilité) du texte source.

1 Table des matières

2 Introduction

Le but de ce document est de présenter l'ensemble des règles retenues par l'Equipe de Développement d'*Aster* (EDA) pour l'écriture des routines FORTRAN du code.

Le respect de ces règles a deux objectifs :

- assurer une bonne portabilité du code,
- assurer une bonne lisibilité (et donc maintenabilité) des sources.

Il est évident qu'il ne suffit pas de respecter ces règles pour atteindre le deuxième objectif. Celui-ci nécessite également des règles de présentation [D9.03.01] et surtout des efforts de la part de chaque programmeur pour se faire comprendre.

> *Il est tout aussi clair que ces règles ne concernent que les aspects du langage utilisé et que d'autres règles doivent être appliquées concernant la programmation (émission des messages d'erreur, utilisation de JEVEUX, usage des structures de données, etc.) ou le développement (présentation, documentation, validation, etc.). Ces règles sont présentées dans le Manuel de Descriptif Informatique et le Manuel d'Administration.*

Notons tout de suite le caractère **impératif** de ces règles. Il ne s'agit pas de conseils vertueux. Chaque règle est écrite de façon à ce que l'on puisse dire sans ambiguïté si elle est respectée ou non : il n'y a rien de qualitatif. Les développeurs du code *Aster* doivent les respecter. Nous verrons que la première règle énoncée (la plus importante) est le respect de la norme ANSI. L'outil de compilation actuel d'*Aster* (`cft77` sur CRAY) permet de vérifier facilement son application. D'autres règles sont vérifiées par l'AGLA [D2.01.02] nous indiquerons systématiquement entre parenthèses le code-retour émis lorsqu'il est non nul (2 ou 4). Le code retour (2) permet à l'administrateur des sources d'*Aster* (ASA) de contrôler les exceptions aux règles. Le code retour (4) interdit la restitution des sources (cf. [§5]). Pour les règles dont la vérification automatique est moins facile, la "sanction" se fera *a posteriori* : la procédure d'évolution des sources permet en effet de retrouver facilement l'identité d'un éventuel développeur négligent.

On ne peut parler de règles de programmation FORTRAN à la DER sans parler du livre "FORTRAN77 Guide pour l'écriture de programmes portables" [bib1] réalisé sous la direction de F. FICHEUX-VAPNE. Ce livre nous a servi de base pour ce document : pratiquement, pour la présentation des règles, nous en avons conservé le plan : éléments lexicaux, ..., entrées-sorties. Nous avons retenu :

- 18 conseils de portabilité,
- 22 conseils méthodologiques.

Le respect de la norme, que nous avons institué en règle n° 1 remplace 33 conseils du livre.

Les conseils du livre ont été érigés en règles, parfois en modifiant leur énoncé : "utiliser ... avec prudence" --> "ne pas utiliser ...".

A ces règles, nous avons ajouté quelques règles qui nous sont propres et qui résultent de l'expérience acquise par l'Equipe de Développement pendant les cinq premières années du Projet.

Dans ce document, nous avons essayé d'expliquer (au moins partiellement) les raisons du choix de ces règles. Ceci n'est pas toujours facile à faire. Pour cela, nous renvoyons à [bib1] pour les règles provenant de ce livre, et nous faisons des renvois à certains paragraphes d'explication pour les règles qui nous sont propres.

Terminons cette introduction en disant que, contrairement à une idée répandue, FORTRAN n'est pas un langage "évident". Certains éléments du langage sont des "survivances" d'anciennes versions du langage. Ces éléments ne sont souvent plus compris des "jeunes" programmeurs. Le lecteur curieux pourra lire avec profit le livre "Fortran 77" de H. KATZAN [bib3] pour bien comprendre ce qu'est le FORTRAN77 de la norme ANSI.

3 Règles retenues

3.1 Présentation

Les règles sont numérotées. Pour les règles issues de [bib1] nous avons conservé la même numérotation que celle du livre, ce qui permet de s'y référer plus facilement, en particulier pour comprendre le pourquoi de cette règle. Les règles supplémentaires que nous nous sommes données sont notées A-i.

A-i : i^{ème} règle *Aster*,
P-i : i^{ème} conseil de portabilité de [bib1],
M-i : i^{ème} conseil de méthodologique de [bib1].

En général, chaque règle est suivie de l'explicitation des exceptions à cette règle (quand elles existent).

rappel :

le code retour de l'AGLA (`asverif`) est écrit entre parenthèses lorsqu'il est non nul : (2) ou (4).

Quelques termes du *Code_Aster*

`JEVEUX :` gestionnaire de mémoire du *Code_Aster*,
`SUPERVISEUR :` "programme principal" qui enchaîne (et supervise) les différentes commandes du logiciel,
`UTILICRA :` bibliothèque contenant les routines non portables (dans leur version "CRAY"),
`BIBC :` bibliothèque contenant quelques utilitaires écrit en C,
`BIBCAL :` bibliothèque contenant quelques routines écrites en "assembleur" CRAY (CAL) : optimisation du gradient conjugué.

3.2 Norme ANSI

A-1 Respecter la norme FORTRAN77 ANSI. (4) [bib2]

C'est évidemment la règle la plus importante.

3.3 Exceptions à la norme

Les exceptions au respect de la norme ANSI sont :

A-2 Utiliser les déclarations `REAL*8` et `COMPLEX*16` (cf. [§4.2]).

A-3 Il est permis de faire `ZK8(I) = ZK8(J)` si `ZK8` est un tableau de chaînes de caractères.

A-4 Il est permis d'utiliser les fonctions hors norme :

- `IAND`, `IOR` sur le type `INTEGER`,
- `DIMAG` et `DCONJG` et `DCMPLX` pour le type `COMPLEX*16` (cf. [§4.3]).

A-5 Les minuscules et certains caractères spéciaux ne sont permis que pour certaines unités de programme (cf. [§4.1]) (2).

A-6 Il est interdit d'utiliser le FORMAT d'impression des variables binaires hors norme Z, B, O, etc… sauf pour la routine `JJIMPO` (2) pour faire le "dump" de la mémoire.

A-7 Il est permis à JEVEUX de mettre en équivalence des variables de type caractère et de type non-caractère (2). (cf. aussi A-13)

A-40 Il est permis de passer en argument une expression de type caractère du style `'chaîne'//arg(1:n)` où `arg` est un argument de longueur inconnue.

Exemple :

```
SUBROUTINE AAAAAA (C)
CHARACTER*(*) C
CALL UTMESS(...,'TOTO'//C(1:4),....)
```

A-41 Il est interdit d'utiliser les constantes données sous forme binaire pour les types non flottants. Exception routine `DEFVEM` (2).

A-44 Il est interdit sauf à JEVEUX (2) de mettre en équivalence un vecteur de longueur 1 avec une variable déclarée en `COMMON`.

3.4 Éléments lexicaux

P.I.6 Inscrire les commentaires dans des lignes commentaires comportant en première colonne un C à l'exclusion de tout autre caractère (4).

A-49 Les lignes blanches sont acceptées.

P.I.8 Ne pas insérer de commentaires entre des lignes suites (0), ni avant les en-têtes `FUNCTION`, `SUBROUTINE`, ni après `END` (4).

M.I.2 Ne pas utiliser de mots-clés comme identificateurs : `IF, END, CALL, GOTO,` …

M.I.3 Ne pas mettre de blancs à l'intérieur des identificateurs, des mots-clés (sauf pour `GO TO`, `END IF`, et `ELSE IF`) et des constantes sauf celles de type caractère.

M.I.5 Ne pas couper en fin de ligne les mots-clés, les identificateurs et les constantes.

A-29 Utiliser le & comme caractère de continuation des cartes suites.

A-30 Couper les lignes à poursuivre de façon à ce que l'instruction soit syntaxiquement incorrecte sans les cartes suites. Par exemple en terminant les lignes à suivre par : `','` `'//'` `'+'` `'-'` `'/'` etc…

A-39 Donner à chaque unité de programme un nom ayant entre 5 et 6 caractères.

3.5 Objets du langage

P.II.3 Ne pas mettre de blancs à l'intérieur d'une constante littérale de type arithmétique ou logique.

P.II.6 Ne comparer que des expressions arithmétiques de même type.

P.II.8 N'utiliser les opérateurs de comparaison `.LT.`, `.LE.`, `.GT.` et `.GE.` sur des chaînes de caractères que pour comparer entre elles des chaînes composées uniquement de chiffres ou des chaînes composées uniquement de lettres.

P.II.9 Utiliser seulement les fonctions intrinsèques `LGE`, `LGT`, `LLE` et `LLT` pour comparer selon l'ordre du code ASCII des chaînes composées à la fois de lettres, de chiffres et de caractères spéciaux.

A-2 (Rappel) Utiliser les déclarations `REAL*8` et `COMPLEX*16`.

A-8 Ne pas utiliser les déclarations `REAL`, `COMPLEX` et `DOUBLE PRECISION` (4).

A-31 Toutes les routines doivent comporter l'instruction (4) :

```
IMPLICIT REAL*8 [A-H] [O-Z] ou
IMPLICIT NONE
```

Ceux qui restitueront du FORTRAN avec `IMPLICIT NONE` seront mieux considérés que les autres.

A-32 Les variables locales non utilisées doivent être détruites.

A-33 Les instructions mises en commentaires doivent être détruites.

A-35 Les étiquettes ne doivent apparaître que devant l'instruction `CONTINUE` ou `FORMAT` (4).

A-36 Utiliser : A**2 à la place de A*A ou de A**2.

3.6 Initialisation - assignation

P.III.1 Utiliser seulement des expressions entières comme valeurs initiale, finale et comme pas d'une boucle `DO` implicite.

M.III.6 Ne pas utiliser l'instruction `ASSIGN`.

A-45 Toutes les constantes numériques (plus petit nombre flottant, etc…) et mathématiques (π, 2π, etc…) doivent être initialisées par appel à une fonction ENVIMA [D6.01.01]. Si celui-ci semble insuffisant émettre un rapport d'anomalie.

A-46 Il est inutile et peu lisible d'utiliser des variables pour manipuler des constantes numériques simples. Par exemple, ce qu'il ne faut pas faire :

```
REAL*8 ZERO,UN
DATA ZERO,UN/0.D0,1.D0/
X = ZERO
MU2 = E / (UN+NU)
```

Ce qu'il faut faire :

```
X = 0.D0
MU2 = E/(1.D0+NU)
```

A-47 Toutes les constantes flottantes doivent comporter le `.` et le `D` (4) (cf. [§4.2]).

3.7 Structures de contrôle

P.IV.1 N'utiliser que des expressions de type entier comme paramètres et compteur de boucle.

P.IV.2 Ne pas modifier le compteur de boucle dans le corps de la boucle.

P.IV.7 Ne pas utiliser l'instruction PAUSE.

M.IV.1 Terminer **chaque** boucle DO par une instruction CONTINUE.

P.IV.4 Utiliser systématiquement l'instruction CONTINUE, en particulier comme dernière instruction des structures de contrôle en ayant soin, lorsque ces structures sont imbriquées, d'attribuer un CONTINUE à chacune.

A-9 Ne pas utiliser le GO TO calculé.
Exceptions : routines TE0000, EX0000, EX0100, OPSEXE.

A-9 Ne pas utiliser le GO TO assigné.

A-10 Ne pas utiliser le IF arithmétique.

A-11 Ne pas utiliser l'instruction STOP sauf pour les routines JEFINI, JVFINM, JVVTAM (2).

A-34 Les blocs IF vide sont interdits.

A-38 Les blocs IF et les boucles DO doivent être indentés de deux caractères.

Exemple :

```
       DO 100 I=1,N
         X(I) = 0.D0
100   CONTINUE
C
       IF(Y.LT.0.D0) THEN
         Z=1.D0
       ELSE
         Z=2.D0
       ENDIF
```

3.8 Unités de programme

P.V.4 Ne pas utiliser les fonctions intrinsèques CHAR et ICHAR (4) (cf. A-15).

Exceptions :
- routines de lecture du superviseur : LXSCAN, LXCAPS, LXINIT (2),
- JEVEUX : JEDEBU, JJVERN (2).

P.V.9 N'utiliser qu'un seul type pour les variables d'un COMMON donné. (cf. [§4.4]).

P.V.12 Utiliser l'instruction SAVE à chaque fois que la rémanence est souhaitée.

M.V.3 Ne pas utiliser les fonctions intrinsèques comme arguments de sous-programme.

M.V.8 Terminer une fonction externe par END sans coder de RETURN. Donc ne pas utiliser RETURN (4). Il n'est pas interdit cependant d'utiliser :

```
      GOTO 9999
      ...
9999  CONTINUE
      END
```

M.V.9 Ne pas définir de fonctions externes ayant le même nom que des fonctions intrinsèques.

M.V.12 Ne pas utiliser dans un sous-programme les retours optionnels de sous-programmes.

M.V.13 Ne pas utiliser l'instruction ENTRY (4) sauf ENVIMA, GETVAL et VALXEM (2).

M.V.16 Ne pas utiliser d'arguments de la forme *étiq (cf. M.V.12).

M.V.17 Ne pas utiliser de BLOCK DATA (4).

M.V.20 Ne pas utiliser l'ordre DIMENSION : il est plus simple de déclarer la dimension avec le type (4).

M.V.21 Ne définir qu'un commun pour chaque instruction COMMON et n'utiliser qu'une seule instruction COMMON par commun.

M.V.22 Pour des objets appartenant à un commun, utiliser les mêmes noms dans toutes les unités de programme où celui-ci apparaît.

M.V.23 Ne pas utiliser le commun blanc (commun sans nom).

M.V.27 Utiliser la notation * pour coder la borne supérieure de la dernière dimension d'un tableau utilisé comme argument formel lorsque cette borne est inconnue du sous-programme.

M.V.28 Noter * la longueur d'une chaîne de caractères utilisée comme argument formel (4).

A-12 Ne pas utiliser l'instruction PROGRAM. Exception : routine aster.

A-13 Ne pas utiliser l'instruction EQUIVALENCE (sauf pour les routines JEVEUX d'UTILICRA) (4) (cf. aussi A-7).

A-14 Ne pas utiliser l'instruction INTRINSIC (4).

A-15 N'utiliser que les fonctions intrinsèques autorisées (cf. [§4.3]) (4).

A-21 L'usage du COMMON est réservé à des utilisations très particulières (cf. [§4.4])

A-26 Limiter à 300 le nombre de lignes d'une routine (2).

A-27 Limiter à 15 le nombre d'arguments d'une routine (2).

A-28 Conserver si possible l'identificateur des arguments de la routine appelée dans la routine appelante.

A-42 N'utiliser l'instruction EXTERNAL qu'en cas de nécessité : lorsque la routine "externe" est passée en argument.

A-43 Ne pas faire appel à des bibliothèques externes : NAG, GENERALE, LINPACK... (4).

A-48 Ne faire des appels externes que dans UTILICRA et BIBC (cf. [§6]) ((2) : pour UTILICRA et BIBC) ((4) : pour les autres).

A-50 Ne pas utiliser les BLAS (4) mais leurs "chapeaux" qui se trouvent dans la bibliothèque UTILICRA [§6.1]. Exception : UTILICRA (2).

3.9 Entrées - sorties

P.VI.4 Lors d'un READ ou d'un WRITE, préciser l'unité logique choisie, en paramétrant son numéro (variable entière). Ne pas utiliser l'astérisque.

P.VI.6 La valeur du code retour IOSTAT dépend du calculateur. La norme indique seulement qu'elle sera nulle si tout s'est bien passé, positive s'il y a eu erreur, négative si une fin de fichier a été rencontrée. N'utiliser que cette propriété.

P.VI.7 Utiliser seulement des expressions de type entier pour les valeurs initiale, finale et pour la valeur du pas des boucles DO implicites dans les listes d'entrée-sortie.

P.VI.8 Donner toujours une liste d'entrée-sortie dans les ordres de lecture et d'écriture et ne pas utiliser le format vide.

P.VI.18 Dans une instruction FORMAT, ne pas utiliser les spécifications d'édition, T, TL et TR.

M.VI.2 Ne pas utiliser l'instruction PRINT.

M.VI.5 Ne pas utiliser le descripteur nHh1...hn mais 'h1h2...hn'.

A-16 Ne pas utiliser l'instruction OPEN
Exceptions : routines, jxlir1, jxouvr, spycod, spyerr.

A-17 Ne pas utiliser l'instruction BACKSPACE.

A-18 Ne pas utiliser l'instruction INQUIRE.

A-19 Ne pas utiliser l'instruction CLOSE.
Exceptions : routines spycod, spyerr.

A-20 Ne pas utiliser l'instruction ENDFILE.

3.10 Problèmes d'inter-compilation

A-22 Utiliser le même type et la même longueur pour une variable mise en COMMON dans toutes les routines qui l'utilisent (4).

A-23 Appeler les routines avec le bon nombre d'arguments (4).

A-24 Ne pas appeler une routine avec un argument du type CHARACTER de longueur différente que celui attendu par le programme appelé (4).

Règle pour les routines ayant un argument formel de type CHARACTER

- utiliser la déclaration CHARACTER*(*) (M-V-28),
- recopier si nécessaire les arguments CHARACTER*(*) dans des variables locales.

A-25 Ne pas appeler une routine avec des arguments d'un type différent de celui qu'elle attend. Notamment ne pas appeler une routine avec un argument complexe si celle-ci attend deux arguments réels et inversement (4).

4 Quelques explications

4.1 Alphabet

Certaines routines utilisent des caractères “interdits” (par la norme).

- Le superviseur doit pouvoir lire les caractères autorisés dans le langage de commandes d’*Aster* :
  - minuscules
  - %, &, !, _.
- JEVEUX utilise : $ et &.
- Les routines imprimant des lignes de “commandes UNIX” utilisent : $, I (“pipe” UNIX), \.

Résumé des exceptions concernant l’alphabet :

Soit : N = {caractères autorisés par la norme}
= {AB...Z01...9”blanc”=+-*/(),.’:$}

• caractères autorisés : N	-{$}	+{&}	toutes les routines
• caractères autorisés : N	+{&}		JEVEUX
• caractères autorisés : N	-{$}	+{minuscules, %, &, !, _}	SUPERVISEUR
• caractères autorisés : N	+{I, \}		impressions UNIX

Les commentaires n’utilisent que les caractères autorisés par la norme.

4.2 Déclaration des types flottants

Ce problème est lié à la double exigence suivante :

- permettre des calculs avec une précision jugée raisonnable : 13 chiffres significatifs sur CRAY,
- être portable sur des machines dont le REAL est de longueur 32 bits.

Malheureusement ces exigences ne sont pas compatibles avec la norme.

La “solution” choisie pour ce problème est détaillée dans la note HI-75/94/068/A “Le problème des nombres flottants en FORTRAN77”.

Solution retenue :

- Les seuls ordres de déclaration autorisés sont :

```
IMPLICIT NONE                  ) au
IMPLICIT REAL*8 [A-H] [O-Z]      ) choix
LOGICAL
INTEGER
REAL*8
COMPLEX*16
CHARACTER*…
```

Sinon code retour (4)

- Écrire les constantes en double précision.
  Sinon code retour (4)

```
REAL*8 R1, R2
COMPLEX*16 C1, C2
```

```
R1 = 1.D0/3.D0
C1 = (3.D0, 4.D0)
```

Il est donc interdit d'écrire :

```
R1 = 1./3.
R1 = 1.E3/3.E3
C1 = (0.,1.)
C1 = (0.E0,1.E2)
```

4.3 Fonctions intrinsèques

Les fonctions intrinsèques du langage : sinus, cosinus, racine_carrée, valeur_absolue, ... ont pour la plupart d'entre elles une forme générique (i.e. indépendante du type de leurs arguments). Ce sont ces fonctions génériques qu'il faut utiliser pour assurer la portabilité.

Exemple : fonction racine_carrée

REAL	nom spécifique : SQRT
DOUBLE PRECISION	nom spécifique : DSQRT
COMPLEX	nom spécifique : CSQRT

Nom générique : SQRT

De la même façon, la conversion de type doit se faire de manière générique. Par exemple, la conversion en entier (troncature) :

Spécifique :

```
INT :      REAL                ->   INTEGER
IFIX :     REAL                ->   INTEGER
IDINT :    DOUBLE PRECISION    ->   INTEGER
```

Nom générique : INT

Les fonctions intrinsèques retenues (Cf note HI-75/94/068/A) sont :

- Fonctions de la norme FORTRAN

  Conversions de type :
  INT DBLE

  Fonctions arithmétiques génériques
  AINT ANINT NINT ABS MOD
  SIGN DIM MAX MIN SQRT
  EXP LOG LOG10

  Fonctions trigonométriques génériques
  SIN COS TAN ASIN ACOS
  ATAN ATAN2 SINH COSH TANH

  Fonctions particulières au type character
  CHAR ICHAR LEN INDEX

- Fonctions hors norme à utiliser

  Conversions de type :
  DCMPLX

  Fonctions particulières au type complexe
  DBLE DIMAG DCONJG

  Fonctions particulières au type integer
  IOR IAND

4.4 COMMON

Le gestionnaire de mémoire JEVEUX permet aux routines du code de s'échanger des données structurées en minimisant le nombre des arguments : on transmet le nom de la structure de données.

L'utilisation de COMMON n'est donc pas recommandée dans *Aster*. Il n'est pourtant pas interdit de les utiliser. Cette utilisation doit cependant rester limitée à des "paquetages" de routines bien identifiés. L'usage de COMMON n'est pas de limiter le nombre des arguments passés aux routines. L'utilisation est justifiée :

- pour des raisons de performances et uniquement pour les routines d'utilisation générale (CALCUL, JEVEUX, SUPERVISEUR, lecture des CATALOGUES),
- pour le passage des paramètres aux routines utilisées en EXTERNAL.

L'utilisation même de JEVEUX est impossible sans les "COMMON JEVEUX" [D6.02.01]. Pour ceux-ci, on utilisera toujours la même séquence de déclarations (recopiée de routine en routine) :

```
      C
      C ---  DEBUT DECLARATIONS NORMALISEES JEVEUX ----------------------------
     ----
      C
            CHARACTER*32     JEXNUM,   JEXNOM,   JEXR8, JEXATR
            INTEGER             ZI
            COMMON / IVARJE /   ZI (1)
            REAL*8           ZR
            COMMON / RVARJE /   ZR (1)
            COMPLEX*16          ZC
            COMMON / CVARJE /   ZC (1)
            LOGICAL             ZL
            COMMON / LVARJE /   ZL (1)
            CHARACTER*8         ZK8
            CHARACTER*16              ZK16
            CHARACTER*24                      ZK24
            CHARACTER*32                              ZK32
            CHARACTER*80                                      ZK80
            COMMON / KVARJE /   ZK8(1),   ZK16(1),  ZK24(1),  ZK32(1),
      ZK80(1)
      C
      C ---  FIN DECLARATIONS NORMALISEES JEVEUX ----------------------------
     ----
```

Pour les autres COMMON nous adopterons la règle de dénomination suivante :

COMMON TxyzPP où :

T est un caractère qui indique le type des variables du COMMON :

I	INTEGER
R	REAL*8
L	LOGICAL
C	COMPLEX*16
L	LOGICAL
K	CHARACTER

PP est une paire de caractères qui identifie à la fois un "paquet" de communs et le paquetage des routines qui utilisent ce paquet.

xyz sont trois caractères libres permettant de différencier les communs d'un même paquet.

Exemples de ce qui **pourrait** être fait :

PP = 'JE' : COMMON nécessaires à JEVEUX
PP = 'CA' : COMMON nécessaires à la routine CALCUL,
PP = 'LC' : COMMON nécessaires à la routine NMCOMP,
PP = 'GC' : COMMON nécessaires aux routines du superviseur.

4.5 Entrées - Sorties

Les entrées/sorties du code sont en principe faites par un nombre restreint de routines :

- lecture/écriture sur les bases de données *Aster* (fichiers d'accès direct) : JEVEUX.
- lecture formatée :
  - le superviseur est chargé de la lecture du fichier de commandes,
  - seules les commandes LIRE_XXXX (XXXX = MAILLAGE, FONCTION, ...) et les commandes d'interfaçage PRE_XXXX (XXXX = GIBI, IDEAS, ...) sont autorisées à lire sur des fichiers externes. Pour cela, la seule instruction autorisée est le READ formaté. L'ouverture (OPEN) et la fermeture (CLOSE) des fichiers sont faites par le superviseur.
- écriture formatée :
  - émissions de messages sur les fichiers ERREUR, MESSAGE, RESULTAT : paquetage des routines UTMESS, UTDEBM, ... [D6.04.01],
  - écriture de résultats : commandes IMPR_XXXX (XXXX = RESU, FONCTION, ...),
  - écriture d'informations dans les fichiers RESULTAT ou MESSAGE (mot clé IMPR de certaines commandes).

Résumé des instructions autorisées :

UTMESS(...)	toutes routines.
READ(nfic, fmt)	Commandes LIRE_XXXX
WRITE(nfic, fmt)	Commandes IMPR_XXXX et mot clé IMPR
OPEN/CLOSE,...	SUPERVIS, JEVEUX
accès direct	JEVEUX

5 "Erreurs" détectées par le compilateur du CRAY

Le compilateur du CRAY cft77 [bib4] émet un certain nombre de messages classés en différentes catégories :

- Comment
- Note
- Caution
- Warning
- Error
- ANSI
- ...

Ces messages ont un en-tête standardisé : `cft77-i` où `i` est un numéro pouvant appartenir à [`1-9999`], (tous les numéros ne sont pas attribués !). Seules les "Error" sont "fatales" pour le compilateur, mais les autres messages (que nous appellerons "erreurs" avec des guillemets) peuvent être révélateurs de problèmes de programmation.

L'explication de chacune de ces "erreurs" est donnée dans le document [D2.02.02].

Nous avons donc décidé de traiter en "erreur" certains de ces messages (en autre les messages ANSI).

Lors d'une compilation (par exemple à la restitution de source), il est facile d'"attraper" les messages correspondant à un numéro donné. On donne alors un code retour à la compilation en fonction du niveau maximum des "erreurs" détectées.

La valeur du code retour est celle de la restitution (cf. `asrest` [D1.02.01]) :

0	Tout va bien	la restitution est acceptée.
2	Alarme	la restitution est soumise au visa de l'ASA.
4	Erreur	la restitution est refusée.

Tous les messages de `cft77` conduisent à un code retour 4 sauf :

- code retour 2 :
  - 720 - Règle A-7
  - 726 - Règle A-44
  - 890 - Règle A-5
- code retour 0 :
  - 118 - Règle A-4 et A-15
  - 408 - Règle A-2
  - 881 - Règle A-3
  - 895 - Règle A-40
- code retour 4 :
  - 342 sauf pour `DEFVEM` - Règle A-41
  - 753 sauf pour `JEIMPO` - Règle A-6

6 Appels "externes"

Nous appelons appel "externe", un `call` vers un point d'entrée qui ne soit :

- ni une routine (ou `ENTRY`) FORTRAN ni une fonction C d'*Aster.*
- ni une fonction intrinsèque

Seules les bibliothèques `UTILICRA` et `BIBC` peuvent effectuer des appels "externes".

Rappelons que l'appel à une routine d'une bibliothèque externe (`NAG`, `GENERALE`, `LINPACK`, …) est interdit (A-43).

En définitive un appel "externe" ne peut être que l'appel à :

/ une routine BLAS
/ une fonction spécifique CRAY
/ une routine de `BIBCAL`

6.1 BLAS : SAXPY, SCOPY, SDOT, SNRM2, SSCAL, SPDOT*

Ces routines changent de noms sur les machines "32 bits" : `DAXPY`, …

Règle :

Utiliser les "chapeaux" à ces routines qui se trouvent dans `UTILICRA` :
`R8AXPY, R8COPY, R8DOT, R8NRM2, R8SCAL, R8PDOT`

* `SPDOT` n'est pas un BLAS officiel.

6.2 Routines spécifiques CRAY

- **Accès direct :** `OPENDR, CLOSDR, READDR, WRITDR`
- **"classiques" :** `AND, XOR, SHIFT1, SHIFTR, INTMAX, LOC, STRMOV`
- **Temps :** `CLOCK, DATE, TREMAIN, SECOND`
- **Environnement UNIX :** `GETCWD`, `GETENV`, `GETPID`
- **Mémoire dynamique :** `HPALLOC`, `HPDEALLC`, `HPCHECK`

6.3 BIBC

Toutes les routines peuvent faire des appels aux routines de la bibliothèque `BIBC`. Mais celle-ci ne doit contenir que des utilitaires qui ne peuvent être réalisés en FORTRAN.

6.4 BIBCAL

Seul `BIBC` peut y faire appel.

7 Bibliographie

[1] FORTRAN77 - Guide pour l'écriture de programmes portables sous la direction de F. Ficheux-Vapné - Éditions Eyrolles Collection de le Direction des Études et Recherches d'Électricité de France

[2] Norme FORTRAN77 - ANSI X3.9-1978

[3] FORTRAN77 - Harry Katzan - Van Nostrand Reinhold Company 1978

[4] CF77 Compiling System, Volume 2 : Compiler Message Manual - SR-3072 4.0 1990 - CRAY research, Inc.