Fortran 90/95

Quelles nouveautés ?

Anne-Sophie Mouronval

Laboratoire MSS-MAT

26 avril 2004

MCours.com

Objectifs

- Connaître les **principaux apports** de F90/95
- Identifier les **aspects obsolètes**

⇨ Formations supplémentaires ou ateliers pour ceux qui le souhaiteront ...

Historique

- 1954 -- Projet de création du premier langage symbolique par IBM :
  FORTRAN (Mathematical FORmula TRANslating System).
- 1966 -- Fortran IV (Fortran 66), première norme .
- 1977 -- Fortran V (Fortran 77).
- 1991/1992 -- Norme ANSI Fortran 90
- 1994 -- Premiers compilateurs Fortran 90 Cray et IBM.
- 1999 -- Sur Cray T3E puis IBM RS/6000 Fortran 95
- 2004 ? -- Fortran 2003

Compatibilité F77/F90

- La norme 77 est **totalement incluse** dans la norme 90.

- Quelques comportements différents parmi lesquels :

  - beaucoup plus de fonctions/sous-programmes intrinsèques.

  ⇒ risque d'homonymie avec procédures externes F77.

  ⇒ attribut `EXTERNAL` recommandé pour les procédures externes non intrinsèques.

  - attribut `SAVE` automatiquement donné aux variables initialisées par l'instruction `DATA`

  (en F77 c'était "constructeur dépendant").

Apports de Fortran 90

1. "Format libre".
2. Objets de types dérivés.
3. Blocs `DO--END DO`, `SELECT CASE`, `WHERE`.
4. Extensions tableaux : profil, conformance, manipulation, fonctions intrinsèques.
5. Allocation dynamique de mémoire (`ALLOCATE`).
6. Pointeurs.
7. Procédures internes (`CONTAINS`), modules (`USE`).
8. Arguments : `OPTIONAL`, `INTENT`, `PRESENT`, passage par mot-clé.
9. Bloc interface, surcharge d'opérateurs ...
10. Procédures récursives.
11. Nouvelles fonctions intrinsèques.

Format & syntaxe

Format libre:

- 132 caractères par ligne;
- `!` : commentaires;
- `&` : continuation de ligne;
- `;` : séparateur d'instruction;
- blancs significatifs.

Exemple :

```
a = 0. ; b = 3.
c = 4.0 * sin(a) * cos(b) + &
    log(b)

PRINT*, "Cette ligne se continue &
        &sur la ligne suivante"
```

Nouvelle syntaxe possible pour certains opérateurs logiques (F77 → F90):

```
.LE. → <=     .LT. → <     .EQ. → ==
.GE. → >=     .GT. → >     .NE. → /=
```

Forme générale d'une déclaration

```
type[, liste_attributs ::] liste_objets
```

Différents types :

* real
* integer
* double precision
* complex
* character
* logical
* type

Différents attributs :

`parameter`	constante symbolique
`dimension`	taille d'un tableau
`allocatable`	objet dynamique
`pointer`	objet défini comme pointeur
`target`	objet accessible par pointeur (cible)
`save`	objet statique
`intent`	vocation d'un argument muet
`optional`	argument muet facultatif
`public` ou `private`	visibilité d'un objet défini dans un module
`external` ou `intrinsic`	nature d'une procédure

Forme générale d'une déclaration (ii)

Exemple :

```
integer :: nbre, cumul
real :: x, y, z

integer, save :: compteur
integer, parameter :: n = 5

double precision a(100)  ! Non recommandé
double precision, dimension(100) :: a
complex, dimension(-2:4, 0:5) :: c

real, pointer :: ptr
```

Types dérivés (i)

Structure de données regroupant des données (**composantes**) hétérogènes.

1. Définition des types dérivés :

```
TYPE Personne
   CHARACTER(LEN=20)  :: nom
   INTEGER            :: age
   REAL               :: taille
END TYPE Personne
```

2. Déclaration (attribut `TYPE`) :

```
TYPE(Personne) :: ind1, ind2, ind3, ind4
```

3. Initialisation *via* un **constructeur de structure** (du même nom que le type créé) :

```
ind1 = Personne( 'Durand', 45, 1.70 )
ind2 = Personne( 'Dupont', 30, 1.64 )
```

MCours.com

Types dérivés (ii)

Affectation et manipulation :

```
ind3 = ind2   ! Possible car opérateur =
              ! sur-défini par la norme 90

ind4 = ind2 + ind3 ! Illicite
```

Accès à un champ : l'opérateur "%" :

```
ind1%nom   ! Durand
ind2%age   ! 30
```

Opérations sur les composantes :

```
INTEGER :: somme
somme = ind1%age + ind2%age
```

Structures de contrôle (i) : boucles DO

Principales nouveautés :

1. Possibilité de nommer les boucles

```
[étiquette:] DO [contrôle de boucle]
                bloc
             END DO [étiquette]
```

2. Boucles `DO` sans contrôle de boucle (condition + `EXIT` pour en sortir)

```
do
   read(*, *) nombre
     if (nombre == 0) EXIT
     somme = somme + nombre
end do
```

3. Instruction `CYCLE`
(abandonne le traitement de l'itération courante)

```
do
   read(*, *, iostat=eof)x
   if (eof /= 0) EXIT
   if (x <= 0.) CYCLE
   y = log(x)
end do
```

Structures de contrôle (ii) : SELECT CASE

Exemple

```
integer :: n

........

SELECT CASE(n)

     CASE(0)                    ! n=0
        print *, ' il est nul'

     CASE(1, 2)                 ! n=1 ou 2
        print *, ' il est petit'

     CASE(3:10)                 ! 3 <= n <= 10
         print *, ' il est moyen'

     CASE(11:)                  ! n <= 11
         print *, ' il est grand'

     CASE DEFAULT               ! autres cas
         print *, ' il est négatif'

 END SELECT
```

Tableaux : définitions

Déclaration d'un tableau : attribut `DIMENSION`.

• **Etendue** d'un tableau dans une dimension = nombre d'éléments dans cette dimension.

• **Profil** d'un tableau = **vecteur** dont chaque élément est l'étendue du tableau dans la dimension correspondante.

Deux tableaux sont **conformants** s'ils ont le même profil.

Exemple :

```
real, dimension(-5:4,0:2) :: x
real, dimension(0:9,-1:1) :: y
```

L'étendue des tableaux `x` et `y` est 10 dans la 1ère dimension et 3 dans la 2ème.

Ils ont même profil : le vecteur (/ 10, 3 /),
⇒ ils sont conformants.

Tableaux : initialisation

Possibilité d'initialiser un tableau 1D au moment de sa déclaration ou lors d'une instruction d'affectation au moyen de **constructeur de tableaux**.

• Pour les tableaux de rang supérieur à 1 on utilisera la fonction `RESHAPE` que nous verrons plus loin.

• Constructeur de tableau : vecteur de scalaires dont les valeurs sont encadrées par les caractères `(/` et `/)`.

Exemple :

```
integer, dimension(4) :: t1, t2,
integer               :: i

t1 = (/ 6, 5, 10, 1 /)
t2 = (/ (i*i, i=1,4) /)
```

Rq ; utilisation de boucles implicites possible.

Tableaux : manipulation

F90 permet de manipuler globalement l'ensemble des éléments d'un tableau (en fait, plusieurs opérateurs ont été sur-définis afin d'accepter des objets de type tableau comme opérande).

Important : les tableaux intervenant dans une expression doivent être conformants.

Exemple :

```
real,    dimension(6,7)       :: a, b
real,    dimension(2:7,5:11)  :: c

 b = 1.5 ! possible car un scalaire est
         ! supposé conformant à tout tableau
 c = b
 a = (b * c) + 4.0
```

Remarque :
Ici l'opérateur * est une multiplication terme à terme (différent de MATMUL)

Tableaux : sections

Il est possible de faire référence à une partie d'un tableau appelée "section de tableau" ou "sous-tableau".

- **Sections régulières** : ensemble d'éléments dont les indices forment une progression arithmétique (**notation par triplets**) de la forme :

```
val_init:val_fin:pas
```

- La notation dégénérée ":" correspond à l'étendue de la dimension considérée.

Exemple :

```
integer, dimension(10)  :: a,b
integer, dimension(20)  :: vec
integer                 :: i

a(1:9:2)  = (/ (i,i=1,9,2) /)
a(2:10:2) = (/ (i,i=-1,-9,-2) /)
b(:)      = (/ (i+1,i=1,7), a(1:3) /)
vec(4:13) = a**2 + b**2
```

Tableaux : sections (ii)

La valeur d'une expression tableau est entièrement évaluée avant d'être affectée.

Ainsi, pour inverser un tableau, on pourra écrire :

```
real, dimension(20) :: tab

      tab(:) = tab(20:1:-1)
```

Ce qui n'est pas du tout équivalent à :

```
integer    :: i

       do i=1,20
        tab(i) = tab(21-i)
       end do
```

Remarque :
il est également possible d'accéder à des éléments quelconques d'un tableau par l'intermédiaire d'un vecteur d'indices ("**indexation indirecte**").
On parle alors de **section irrégulière** ...

Tableaux : fonctions intrinsèques (i)

Interrogation

`SHAPE(array)` et `SIZE(array[,dim])` :
profil et taille (ou étendue de la dimension indiquée *via* `dim`) de `array`.

`UBOUND(array[,dim])` et `LBOUND(array[,dim])` :
bornes sup/inf de chacune des dimensions (ou seulement de celle indiquée *via* `dim`) de `array`.

`MAXVAL(array[,dim][,mask])` et
`MINVAL(array[,dim][,mask])`
où `mask` est un tableau de type logical conformant avec `array`.
max et min de `array`.

Exemple :

```
          | 1  3  5 |
    A =   |         |
          | 2  4  6 |
```

`MINVAL(A,dim=2,mask=A>3)` retourne `(/ 5,4 /)`.

Autres fonctions : `MAXLOC(…)`, `MINLOC(…)`,
`ALL(…)`, `ANY(…)`, `COUNT(…)`.

Tableaux : fonctions intrinsèques (ii)

Construction/transformation

```
RESHAPE(source,shape[,pad][,order])
```

Cette fonction permet de construire un tableau d'un profil donné à partir d'éléments d'un autre tableau.

Exemple :

```
RESHAPE((/ (i,i=1,6) /), (/ 2,3 /))
```

a pour résultat :

```
| 1  3  5 |
| 2  4  6 |
```

Autres fonctions :

```
CSHIFT(...), EOSHIFT(...),
PACK(...),   UNPACK(...),
SPREAD(...), MERGE(...),
TRANSPOS(...).
```

Tableaux : fonctions intrinsèques (iii)

`PRODUCT(array[,dim][,mask])` et
`SUM(array[,dim][,mask])` :
somme et produit des éléments d'un tableau.

Exemple :

```
         | 1   3   5 |
A =      |           |
         | 2   4   6 |
```

`SUM(A,dim=1)` retourne `(/ 3,7,11 /)`.

`DOT_PRODUCT(vector_a,vector_b)` :
produit scalaire des deux vecteurs.

`MATMUL(matrix_a,matrix_b)` :
produit matriciel de deux matrices ou d'une matrice et d'un vecteur.

Tableaux : instruction et bloc WHERE

L'instruction `WHERE` permet d'effectuer des affectations de type tableau par l'intermédiaire d'un filtre (masque logique).

Forme générale :

```
WHERE (mask)
          bloc1
ELSEWHERE
          bloc2
END WHERE
```

où `mask` est une expression logique retournant un tableau de logiques.

Exemple :

```
real, dimension(10) :: a

    WHERE (a > 0.)
           a = log(a)
    ELSEWHERE
    a = 1.
    END WHERE
```

Tableaux dynamiques (i)

- En Fortran 90, il est possible de faire de l'**allocation dynamique de mémoire**.
- Les tableaux ainsi crées sont appelés "**tableaux dynamiques**" ou "**tableaux à profil différé**.
- Déclaration : spécifier l'attribut `ALLOCATABLE`.
- Allocation *via* l'instruction `ALLOCATE` à laquelle on indique le profil désiré.
- Libération de l'espace mémoire allouée : instruction `DEALLOCATE`.
- La fonction intrinsèque `ALLOCATED` permet d'interroger le système pour savoir si un tableau est alloué ou non.

Tableaux dynamiques (ii)

Exemple :

```
real, dimension(:,:), ALLOCATABLE :: a
Integer  :: n,m,err

read *, n, m

   if (.not. ALLOCATED(a)) then

        ALLOCATE(a(n,m),stat=err)

        if (err /= 0) then
        print *, "Erreur à l'allocation &
                 &du tableau a"
        stop 4
        end if

   end if

DEALLOCATE(a)
```

Pointeurs (i)

• Définition

En C, Pascal ⇒ variable contenant l'adresse d'objets.

En Fortran 90 ⇒ **alias.**

• **États d'un pointeur**

1. **Indéfini** : à sa déclaration en tête de programme

2. **Nul** : alias d'aucun objet

3. **Associé** : alias d'un objet appelé cible.

MCours.com

Pointeur (ii)

- **Cible** : objet pointé (attribut `TARGET`).

- Le **symbole binaire** => sert à valoriser un pointeur.

Exemple :

```
integer, target  :: n
integer, pointer :: ptr1, ptr2

n = 10
ptr1 => n
ptr2 => ptr1
n = 20
print *, ptr2 ! on trouve 20
```

Remarques :

1. `p1` et `p2` étant deux pointeurs, `p1 => p2` implique que `p1` prend l'état de `p2` (indéfini, nul ou associé à la même cible)

2. La fonction intrinsèque `ASSOCIATED` permet de comparer deux pointeurs

Procédures (i)

F77 : procédures externes (= **unité de compilation**)

F90 : nouvelles possibilités :

1. **Procédure interne** :

procédure contenue à l'intérieur d'un programme principal, d'une procédure externe ou d'une procédure-module. Elle

- est appelée de la même façon qu'une procédure externe
- est compilée en même temps que l'unité qui l'héberge (son **hôte**)
- a accès à des informations appartenant à son hôte
- est placée après le mot clé `CONTAINS`
- n'est pas visible de l'extérieur de son hôte.

2. **Procédure-module** :

procédure contenue dans un module après le mot clé `CONTAINS`. Elle peut contenir une procédure interne.

Procédure interne : exemple

```
PROGRAM Thingy

    IMPLICIT NONE
    .....
    CALL OutputFigures(NumberSet)
    .....

    CONTAINS

     SUBROUTINE OutputFigures(Num)
     REAL, DIMENSION(:), INTENT(IN) :: Num
     PRINT*, "Here are the figures", Num
     END SUBROUTINE OutputFigures

END PROGRAM Thingy
```

Remarque :

Ne pas confondre les procédures **internes/externes** (lieu d'apparition de la procédure) et les procédures **intrinsèques/"extrinsèques"** (provenance de la procédure).

Procédure : interface

En F77 : procédures externes, **interface inexistante.**

⇒ Risque d'incohérence entre l'utilisation de cette procédure et sa déclaration locale

(erreurs non détectées à la compilation : nombre d'arguments incorrect *etc* ...).

Solutions en Fortran 90 :

1/ par le biais de procédures internes ou de procédures-modules (**interfaçage automatique**)

2/ par le biais de **blocs-interfaces explicites** présents ou accessibles permettant de préciser à l'unité appelante toutes les contraintes d'appel.

⇒ Les erreurs de cohérence seront détectées à la compilation.

Remarque :

Pour les fonctions intrinsèques, l'interface est implicite.

Interface explicite "simple" (ou anonyme)

Subroutine externe maxmin :

```
Subroutine maxmin(vect,v_max,v_min,ctl,rgmax)
 implicit none
 real,dimension(:),intent(in)     :: vect
 real,             intent(out)    :: v_max,v_min
 integer,optional, intent(out)    :: rgmax
 integer,          intent(inout)  :: ctl

 v_max = MAXVAL(vect) ; v_min = MINVAL(vect)
 ctl = 1; …

end subroutine maxmin
```

Interface à placer dans chaque unité appelante :

```
!------------ Bloc interface----------------
interface
 Subroutine maxmin(vect,v_max,v_min,ctl,rgmax)
  real,dimension(:), intent(in)     :: vect
  real,              intent(out)    :: v_max,v_min
  integer, optional, intent(out)    :: rgmax
  integer,           intent(inout)  :: ctl
 end subroutine maxmin
end interface
!-----------------------------------------
```

Apports des interfaces

- Détection des erreurs liées à la non cohérence des arguments d'appel et des arguments muets (type, attributs et nombre).

- Possibilité de contrôler la **vocation** des arguments en fonction des attributs `INTENT` et `OPTIONAL`.

- Possibilité de tester l'absence des arguments optionnels (fonction `PRESENT`).

- Passage d'arguments par mot-clé.

- Transmission du profil et de la taille des tableaux à **profil implicite** et possibilité de les récupérer *via* les fonctions `SHAPE` et `SIZE`.

Attributs INTENT et OPTIONAL, passage par mot-clé

Préciser la **vocation des arguments** muets de façon à pouvoir contrôler plus finement l'usage qui en est fait.

Pour ce faire, F90 a prévu :

1. l'attribut `INTENT` d'un argument :

- o entrée seulement ⇒`INTENT(IN)`,
- o sortie seulement ⇒ `INTENT(OUT)`,
- o mixte ⇒ `INTENT(INOUT)`

2. l'attribut `OPTIONAL` pour déclarer certains **arguments** comme **optionnels** et pouvoir tester leur présence éventuelle dans la liste des arguments d'appel (fonction intrinsèque `PRESENT`).

Par ailleurs, F90 permet aussi le **passage des arguments par mot-clé** (+ fiable que le **passage positionnel**).

Exemple (i)

On considère la subroutine externe suivante :

```
Subroutine maxmin(vect,v_max,v_min,ctl,rgmax)

 implicit none
 real,dimension(:),intent(in)     :: vect
 real,             intent(out)    :: v_max,v_min
 integer,optional, intent(out)    :: rgmax
 integer,          intent(inout)  :: ctl

 v_max = MAXVAL(vect)
 v_min = MINVAL(vect)
 ctl = 1

 if(present(rgmax)) then
   rgmax = MAXLOC(vect, DIM=1)
   ctl = 2
 endif

 print *,'Taille vecteur :',SIZE(vect)
 print *,'Profil vecteur :',SHAPE(vect)

end subroutine maxmin
```

Exemple (ii)

```
Program inout

 implicit none
 integer,parameter    :: n=5
 integer              :: rgmax=0,ctl=0
 real,dimension(n)    :: v=(/ 1.,2.,40.,3.,4. /)
 real                 :: vmax,vmin

!------------ Bloc interface----------------
Interface
Subroutine maxmin(vect,v_max,v_min,ctl,rgmax)
  real,dimension(:), intent(in)     :: vect
  real,              intent(out)    :: v_max,v_min
  integer, optional, intent(out)    :: rgmax
  integer,           intent(inout)  :: ctl
 end subroutine maxmin
end interface
!-----------------------------------------

!-- Appel positionnel
 call maxmin(v, vmax, vmin, ctl, rgmax)
!-- Appel positionnel ss arg. optionnel rgmax
 call maxmin(v, vmax, vmin, ctl)
!-- Idem a mot cle
 call maxmin(vect=v, v_max=vmax, &
                     ctl=ctl, v_min=vmin)
  .......

end program inout
```

Tableaux à profil implicite

Extrait du programme appelant :

```
…
integer, dimension(5,6) :: t
  interface      !<<<<<<<<<<<<<<<<<<<<|
    subroutine sp(t)                  |
      integer, dimension(:,:) :: t    |
    end subroutine sp                 |
  end interface !<<<<<<<<<<<<<<<<<<<<|

t = 0 ; call sp(t) …
```

Subroutine externe :

```
subroutine sp(t)
 integer, dimension(:,:) :: t
 integer                 :: i
 print *,size(t,1), size(t,2)
   do i=1, size(t,1)
    print *,(t(i,j), j=1,size(t,2))
   end do
end subroutine sp
```

`t` est un tableau à profil implicite.

Rq : ne pas confondre **"tableau à profil implicite"** (recommandé) et **"tableau à taille implicite"** (déconseillé).

Cas d'interface "explicite" obligatoire

Il existe **10 cas** où une **interface** est **obligatoire**, parmi lesquels :

- fonction à valeur tableau,
- fonction à valeur pointeur,
- tableau à profil implicite,
- passage d'arguments à mots-clé,
- argument optionnel,
- procédure générique,
- surcharge ou définition d'un opérateur,
- surcharge du symbole d'affectation.

...

Interfaces génériques (i)

En + des interfaces "simples", il existe des interfaces "**génériques**" (3 types).

1. Les **interfaces "nommées"** (ou génériques *stricto sensu*)

Exemple (à placer dans un module) :

```
INTERFACE CLEAR    !  Nom fct generique
 MODULE PROCEDURE clear_int
 MODULE PROCEDURE clear_real
END INTERFACE
```

+ définition des procédures `clear_int` et `clear_real`

2. Les **interfaces opérateurs** (création ou sur-définition d'opérateurs)

Exemples (à placer dans un module) :

```
INTERFACE OPERATOR (.TWIDDLE.)
 MODULE PROCEDURE itwiddle, rtwiddle
END INTERFACE

INTERFACE OPERATOR (*)
 MODULE PROCEDURE mult
END INTERFACE
```

+ définition des procédures

Interfaces génériques (ii)

3. Les **interfaces-assignations** (pour surcharger le symbole d'affectation =)

Exemple (à placer dans un module) :

```
INTERFACE ASSIGNMENT (=)
 MODULE PROCEDURE real_to_int
END INTERFACE
```

+ définition de la fonction `real_to_int`

MODULES

• Un **module** est une **unité de programme** particulière (mot clé **MODULE**) introduite en Fortran 90 pour **encapsuler** :

- des données et des définitions de types dérivés,
- des blocs interfaces,
- des procédures (après l'instruction `CONTAINS`),

...

• Son utilisation au sein d'un programme se fait grâce à l'instruction `USE` suivi du nom du module [+ mot clé `ONLY`].

• Ces ressources sont par défaut accessibles à l'utilisateur (attribut `PUBLIC`) mais peuvent être rendues privées (attribut `PRIVATE`).

• Il doit être compilé séparément avant de pouvoir être utilisé.

Exemple

Module (dans un fichier sépare de préférence) :

```
MODULE stack

IMPLICIT NONE
INTEGER, PARAMETER :: stack_size = 100
INTEGER, SAVE :: store(stack_size), pos= 0
    CONTAINS
  SUBROUTINE push(i)
    …
  END SUBROUTINE push

END MODULE stack
```

Programme utilisant ce module :

```
PROGRAM StackUser
USE stack  ! Place avant tout le reste

IMPLICIT NONE
       …
      CALL push(14); CALL push(21);

       …
END PROGRAM StackUser
```

MCours.com

Quelques fonctions intrinsèques utiles (i)

- Les types prédéfinis en F90 sont en fait des noms génériques renfermant chacun un certain nombre de **variantes** ou **sous-types** que l'on peut **sélectionner** à l'aide du **paramètre** `KIND` lors de la déclaration des objets.

- Fonction intrinsèque `SELECTED INT_KIND(r)`

Elle retourne un entier qui correspond au sous-type permettant de représenter les entiers n tels que :

$$-10^r < n < 10^r$$

- Fonction intrinsèque `SELECTED_REAL_KIND(p,r)`

(`p`=précision et `r`=étendue).

Elle retourne un entier qui correspondant au sous-type permettant de représenter les réels x répondant à la demande avec :

$$10^{-r} < |x| < 10^{+r}$$

Exemple

```
! On decide que les entiers n,m sont < 99
  integer, parameter      ::  p = selected_int_kind(2)
  integer(kind=p)         ::  n, m
```

Quelques fonctions intrinsèques utiles (ii)

`TINY(x)` : plus petite valeur réelle représentable dans le sous-type de `x` (limite d'underflow).

`HUGE(x)` : plus grande valeur réelle ou entière représentable dans le sous-type de `x` (limite d'overflow).

Aspects obsolètes en F90

1. `IF` arithmétique : `IF (ITEST) 10,11,12`

2. Branchement au `END IF` depuis l'extérieur (H.N.95)

3. Boucles `DO` pilotées par réels (H.N.95)

4. Partage d'une instruction de fin de boucle

5. Fins de boucles autres que `CONTINUE` ou `END DO`

6. `ASSIGN` et le `GO TO` assigné (H.N.95)

7. `ASSIGN` d'une étiquette de `FORMAT` (H.N.95)

8. `RETURN` multiples

9. `PAUSE` (H.N.95)

10. Format d' édition `Hn` (H.N.95)

(H.N.95) : aspect devenant Hors Norme 95 .

Principales nouveautés de la norme 95

1. Bloc `FORALL`
2. Attributs `PURE` et `ELEMENTAL` pour certaines procédures
3. Fonction intrinsèque `NULL()` pour forcer un pointeur à l'état nul
4. Valeur initiale par défaut pour les composantes d'un type dérivé
5. Fonction intrinsèque `CPU_TIME`
6. Bloc `WHERE` : imbrication possible
7. Expressions d'initialisation étendues

8. `MAXLOC/MINLOC` : ajout de l'argument `dim`

Etc …

Futur = Fortran 2003 (+ orienté objet : héritage …)

Aspects obsolètes en F95

1. Le "format fixe" du source
   ⇒ "format libre".

2. Le `GO TO` calculé
   ⇒ `SELECT CASE`.

3. L'instruction `DATA` placée au sein des instructions exécutables
   ⇒ avant les instructions exécutables.

4. *Statement functions*
(`sin_deg(x)=sin(x*3.14/180.)`)
   ⇒ procédures internes.

5. Le type `CHARACTER*...` dans les déclarations
   ⇒ `CHARACTER(LEN=....)`.

6. Le type `CHARACTER(LEN=*)` de longueur implicite en retour d'une fonction
   ⇒ `CHARACTER(LEN=len(str))`.

Conseils pour passer de F77 à F90

1. S'assurer de la fiabilité du code de départ
(passer en **IMPLICIT NONE** …)

2. Passer en format libre

3. Eliminer les aspects obsolètes de F77
(détection à la compilation grâce à des options, exemple pour xlf :
`qlanglvl=90std/95std/95pure/…`)

4. Créer des modules, mettre en oeuvre des interfaces (réfléchir au regroupement des sous-programmes, à l'arborescence des modules …)

5. Retoucher les listes d'arguments
(préciser les vocations, tableaux à profil implicite, passage par mots-clé)

6. Utiliser les nouvelles possibilités pour les tableaux et les fonctions intrinsèques

Remarque : il existe des outils d'aide à la conversion (ex NEC : Trans90 de PSUITE ...)

Références

A la bibliothèque ECP

• DELANNOY C. *Programmer en Fortran 90 Guide complet* (1997)

• DUBESSET M., VIGNES J. *Les Spécificités du Fortran 90* (1990)

•LIGNELET P. *Manuel complet du langage Fortran 90 et Fortran 95* (1996)

• LIGNELET P. *Structures de données et leurs algorithmes avec Fortran 90 et Fortran 95* (1996)

• METCALF M., REID J.,*Fortran 90/95 explained* (1996)

Sur Internet

• www.liv.ac.uk/HPC/HTMLFrontPageF90.html

• www.idris.fr dans les supports de cours

MCours.com