Bernard PICHON (*)
Département Cassiopée à Nice
UMR 6202 du CNRS
Observatoire de la Côte d'Azur
BP 4229 , 06304 Nice Cedex 04

Courriel : Bernard.Pichon@obs-nice.fr
Courriel : Bernard.Pichon@obs-azur.fr
Courriel : Bernard.Pichon@oca.eu

Disponible à : http://www.obs-nice.fr/pichon/cours.html
Disponible à : http://www.oca.eu/pichon/cours.html

FORTRAN 90/95 : QUELLES NOUVEAUTÉS ?

Nouvelle Norme ISO :

Langage de base :1539-1:1996 plus TR 15580 et TR 15581
aussi (maintenant) 1539-1:2004 plus (bientôt) un TS (co-array)

Mais aussi des Normes collatérales :

1539-2 dite ISO_VARYING_STRING
1539-3 sur le préprocesseur Fortran dit aussi CoCo

Norme vivante : Caractéristiques *dépréciées*, obsolètes ou détruites (F95,f2k)

Fortran 2003 (f2k) : quelques compilateurs en cours d'élaboration

Remarque : Disponibilité de (*bons*) compilateurs, de bibliothèques

(*) : Membre du Groupe d'Experts (GE5) de l'AFNOR/CGTI/CN22 (Fini !)
Un des 3 membres représentant la France (et le seul du CNRS !)
du groupe de travail de l'ISO-IEC/JTC1/SC22/WG5 (Fortran)

I-a : F90/95 ET LE CONFORT D'ÉCRITURE

Format libre (jusqu'à 132 caractères par ligne !)

Caractère de continuation en fin de ligne : `&`

Caractère de commentaire (de fin et de début de ligne) : `!`

Rôle significatif du blanc

Délimiteurs de chaînes de caractères : `'` ou `"`
pour un *seul* caractère : `'a'`
pour une chaîne : `"C'est l'été"`

Noms de variables pouvant atteindre 31 caractères (dont _)

Les minuscules font partie du standard et sont identiques aux majuscules

```
Double Precision :: SpeedOfLight
```

Séparateurs d'instructions : `;`

```
A = 0 ; B = 0 ; C = 0
```

Écriture des opérateurs logiques : `< , > , <= , >= , == , /=`

Directive `INCLUDE`

Quelques caractères d'usage *réservé* (voir par la suite) : `(/ , /) , % , ::`
voir …. `[ ]`

Questions de vocabulaire (voir par la suite) :
Constantes littérales et constantes nommées, Unités de stockage,
Portée (unité de --), Unité hôte, Fonction d'interrogation

I-b : F90/95 ET LE CONFORT DE PROGRAMMATION

Instruction : `Implicit None`

Possibilité de nommer l'instruction END associée à une unité de programme :

```
Program PREMIER                 Subroutine SUB1
.                               .
.                               .
.                               .
End Program PREMIER             End Subroutine SUB1
```

Possibilité de donner des noms aux constructions IF et DO :

```
Test1 : If (...) Then       Suite : Do i = 1 , 10
        .                           .
        Else                        .
        .                           .
        .                           End Do Suite
        End If Test1
```

Possibilité bien plus grande pour définir des constantes nommées,
y compris pour des tableaux :

```
Real, Dimension(100), Parameter :: ...
```

Liste d'attributs par objet et initialisation des variables :

```
Integer :: count = 0
Real, Dimension(0:9) :: A
Real, Dimension(10,10), Save :: B
```

Remarque : Attribut `Save` (*très*) recommandé

Constructions DO :

DO , END DO , CYCLE et EXIT

```
B1 : Do
        ...
        If (...) CYCLE B1
        ...
B2 :    Do
           ...
           If (...) CYCLE B1
           ...
           If (...) EXIT B2
           ...
        End Do B2
        ...
        If (...) EXIT B1
        ...
     End Do B1
```

Construction CASE :

SELECT CASE , END SELECT , CASE et CASE DEFAULT

```
Select Case (I)
   Case (0)
      ...
   Case (1,3)
      ...
   Case (2,4:6)
      ...
   Case (7:)
      ...
   Case Default
      ...
End Select
```

```
Select Case (rep)
   Case('y', 'Y', 'o', 'O')
      ...
   Case('n', 'N')
      ...
   Case Default
      ... ! mauvaire
      ... !  reponse
 End Select
```

Pour mémoire (voir par la suite) : WHERE et FORALL

Remarque : On peut reconstruire DO WHILE avec EXIT !

Amélioration des E/S avec des fonctionnalités supplémentaires :

E/S par listes nommées : `NAMELIST`
`Advance="No"`
`Status="Replace"`
`Action="Read" , "Write" , "ReadWrite"`
`Position="Rewind" , "Append" , "AsIs"`
voir aussi : `Blank= , Delim= , Pad=`

Nouvelle instruction du type 'Inquire' : Instruction `Inquire` par liste

Amélioration de quelques formats :

Rappel (F77) : formats dynamiques, descripteur `':'`
Descripteur d'édition pour des entiers : `B , O , Z`
Descripteur d'édition pour des réels : `EN , ES`
Descripteur de contrôle : `TR , TL` et aussi `BN` et `BZ`
Format de sortie minimal (F95) *e.g.* `I0`

Quelques fonctions intrinsèques supplémentaires :

pour des nombres :
`CEILING` et `FLOOR` (*vs.* `INT` et `NINT`)
`MODULO` (*vs.* `MOD`)

pour des caractères :
`ACHAR` et `IACHAR` (*vs.* `CHAR` et `ICHAR`)

pour des chaînes de caractères :
`LEN_TRIM` et `TRIM`
`ADJUSTL` et `ADJUSTR`
aussi : `VERIFY , SCAN , REPEAT`

pour des appels systèmes :
`RANDOM_NUMBER , RANDOM_SEED`
`DATE_AND_TIME , SYSTEM_CLOCK , CPU_TIME` (F95)

pour des manipulation de bits :
`BTEST , IBCLR , IBSET`
`IAND , IEOR , IOR , NOT`
`ISHIFT , ISHITC`
aussi `BIT_SIZE , IBITS , MVBITS`

pour manipuler directement dans la mémoire (cf. `EQUIVALENCE`) :
`TRANSFER`

II : LA GRANDE NOUVEAUTÉ DE F90 :

LE TRAITEMENT DES TABLEAUX

Les procédures élémentaires :

Les fonctions (mathématiques) de base sont devenues élémentaires
On peut aussi créer des fonctions (F95) :
parallèlisables (`PURE`), élémentaires (`ELEMENTAL`)

Vocabulaire : Rang, Étendue, Profil, Tableaux vides, Tableaux conformes

	Rang	Étendue	Profil
Scalaire	0	--	--
Vecteur	1	n	(/ n /)
Matrice	2	n *ou* m	(/ n , m /)

Exemples :
```
Real, Dimension(10) :: A, Y
Real, Dimension(10,10) :: M
```

Constantes de type tableau, Constructeurs de vecteurs :

```
Integer, Dimension(3),Parameter :: IJ=(/ 1,4,3 /)
...
A = (/ (REAL(i),i=1,10) /)
```

Sections de tableaux (*i.e.* sous-tableaux) y compris adressage indirect :

```
A(:5)     M(1,:)     A(IJ)
```

ATTENTION : ne pas utiliser de sections (`:`) de tableaux si ce n'est pas voulu !!

Calculs sur des tableaux complets ou des sections :

```
A = 2. * Y + 1.
A(:5) = SIN( M(1,1:9:2) )
A(3:10) = A(1:8)
M(:,(/i,j/))) = M(:,(/j,i/))
```

Allocation dynamique :

Attribut `ALLOCATABLE`
Instructions `ALLOCATE` et `DEALLOCATE`
Fonction d'interrogation `ALLOCATED`

```
Real, Dimension(:), Allocatable :: AA
...
ALLOCATE ( AA(0:N) , Stat=ios )
...
If (ALLOCATED(AA))    &           ! i.e. de 0 a N
     AA = (/ (REAL(i),i=LBOUND(AA),UBOUND(AA)) /)
...
DEALLOCATE ( AA )
```

Objets automatiques : Très utile pour des tableaux de travail (***Attention à la pile***)

```
Subroutine SWAP(A1,A2)
   Real, Dimension(:) :: A1, A2
   Real, Dimension(Size(A1)) :: W
   W = A1 ; A1 = A2 ; A2 = W
End Subroutine Swap
```

Passage de tableaux en arguments : utilisation des modules (*cf. infra*) MAIS …

```
Module MOD_SUB
   Implicit None
Contains
   !
   Subroutine SVOIR ( SVECT )
      Real, Dimension(:), Intent(In) :: SVECT
      Write(*,*) LBOUND(SVECT), UBOUND(SVECT)
   End Subroutine SVOIR
   !
   Subroutine AVOIR ( AVECT )
      Real, Dimension(:), Allocatable, Intent(In) :: AVECT
      Write(*,*) LBOUND(AVECT), UBOUND(AVECT)
   End Subroutine AVOIR
   !
End Module MOD_SUB
!!!!
Program COLON
   USE MOD_SUB
   Implicit None
   Real, Dimension(:), Allocatable :: ATAB
   Integer :: ios
   !
   Allocate( ATAB(-10:+10) , Stat=ios) ; If ( ios /= 0 ) STOP " 1 "
   !
   Write(*,*) LBOUND(ATAB), UBOUND(ATAB)
   Call SVOIR ( ATAB )
   Call AVOIR ( ATAB )
   Call AVOIR ( ATAB(:) )     ! pas la meme chose du tout !!!
   !
   STOP " "
End Program COLON
```

Procédures intrinsèques pour utiliser des tableaux :

Concernant les caractéristiques des tableaux :

```
LBOUND , UBOUND , SIZE , SHAPE
```

Concernant les éléments extrémaux des tableaux :

```
MINVAL , MAXVAL , MINLOC , MAXLOC
```

Pour la somme et le produit des éléments de tableaux :

```
SUM , PRODUCT
```

Il y a aussi les arguments optionnels : `DIM` et `MASK` (**très** utile)

```
SUM ( M , Dim=1 , Mask=BB>0. )
```

Pour l'algèbre linéaire :

```
DOT_PRODUCT , MATMUL , TRANSPOSE
```

Pour la 'transformation' des tableaux (et aussi l'initialisation) :

```
RESHAPE
```

Pour des tableaux logiques :

```
ALL , ANY , COUNT
```

Pour des opérations de décalage :

```
CSHIFT , EOSHIFT
```

Aussi :

```
MERGE , PACK , UNPACK , SPREAD
```

Il faut l'utiliser pour y croire !

Moralité : Presque plus besoin d'écrire de boucles DO et cela, sans faire d'erreurs, par exemple, sur les bornes ou les dimensions des tableaux (40% des programmes sont faux pour cette unique raison).

Optimisation (de la mémoire, de la pile) : ATTENTION pour de grands tableaux !

```
Double Precision, Dimension(ndim,ndim) :: A, B, C

A = MATMUL( B , C )

Do j = 1 , ndim
   Do k = 1 , ndim
      Do i = 1 , ndim
         A(i,j) = A(i,j) - B(i,k) * C(k,j)
      End Do
   End Do
End Do
```

Voir, par exemple, BLAS-3 (bug/problème à contrôler dans xGEMM)

Pire (!) : `A = MATMUL( B , MATMUL(A,TRANSPOSE(B)) )`

Pour commencer à faire de la parallélisation :

Construction WHERE (éventuellement imbriquées en F95) :

Instructions `WHERE` , `ELSEWHERE` , `END WHERE`

```
Where ( A > 0.)       ! plot lin-log
   Y = LOG10(A)
ElseWhere
   Y = -30.           ! hors cadre
End Where
```

Construction FORALL (F95) : Instructions `FORALL` et `END FORALL`

```
ForAll ( I=1:10 ) A(I) = M(I,I)
...
ForAll ( I=1:10 , J=1:10 )
   M(I,J) = 1. / REAL(I+J)
End ForAll
```

Remarques : On peut imbriquer des instructions `FORALL` et `WHERE`
On peut aussi ajouter un masque dans l'instruction `FORALL`

Il existe aussi des fonctions à valeur_de_tableau mais en F95, l'utilisation de l'attribut `ELEMENTAL` est conseillé !

III-a : F90/95 ET LE GÉNIE LOGICIEL

Arguments nommés (nom-clé d'arguments) et arguments optionnels :

Attribut `OPTIONAL`

Fonction d'interrogation `PRESENT`

Mode d'association des arguments (vocation) :

Attribut (ou instruction) `INTENT` de valeur `In` , `Out` ou `InOut`

Interfaces explicites, visibles, implicites (USE), sans objet (F77) :

Déclaration `INTERFACE` et `END INTERFACE`

```
zero = CALCZERO (Fonc=Degre3, xinit=x0)
...
zero = CALCZERO (Degre3, xa=x0-1., xb=x0+1.)

Real Function CALCZERO (FONC,xinit,xa,xb) Result (zero)
   Interface
      Real Function FONC (x)
         Real, Intent(In) :: x
      End Function FONC
   End Interface
   Real,Optional,Intent(In) :: xinit, xa, xb
   !
   Logical :: Newton, Dicho
   !
   Newton = PRESENT(xinit)
   Dicho = PRESENT(xa) .AND. PRESENT(xb)
   !
   If ( Newton .AND. .NOT.Dicho ) Then
      Call CALC_NEWTON (FONC, xinit, zero ) ; Return
   End If
   !
   If ( Dicho .AND. NOT. Newton ) Then
      Call CALC_DICHO ( FONC, xa, xb, zero ) ; Return
   End If
   !
   Write(*,*)" Usage non conforme "
   zero = HUGE(zero)
   !
End Function CALCZERO
```

La déclaration explicite d'interface est **très utile** pour la déclaration d'arguments de type procédure et **obligatoire** pour la surcharge de fonction ou d'opérateurs.

Création de module :

Instructions `MODULE` et `END MODULE`

Importation de ressources avec `USE` avec, évt., la clause `ONLY`

- Cas de données seulement *e.g.* variables (remplacement des commons)

```
Module PRECISION
   Integer, Parameter :: SP = KIND(1.) ,   &
                         DP = KIND(1.D0)
End Module PRECISION
...
Program TEST
   USE PRECISION , P => DP
   Real(P), Parameter :: Pi = 3.14159265358979323846_P
 ! Real(P), Parameter :: Pi = 4.0_P * ATAN(1.0_P) ! f2k
End Program TEST
```

- Cas de procédures : création automatiques d'interfaces implicites

  Ce qui permet une meilleure utilisation des tableaux en arguments

  ou Procédures internes (on ne parle plus de fonction-instruction !) :

```
CONTAINS
```

  Voir l'exemple précédent sur les bornes d'un tableau (passage par descripteur).

- **Les deux :** on commence à penser POO …. (voir plus loin)

Aussi :

Clause `RESULT` :

Utile pour la lisibilité des fonctions et nécessaire pour les fonctions à valeur_de_tableau et en cas de récursivité.

Il existe aussi des pointeurs : Attribut `POINTER` et `TARGET`

Remarque : `DOUBLE(qqch)` est équivalent à : `REAL(qqch,DP)`

III-b : Compléments F90/95

- Différents types numériques possibles (aussi pour les caractères) :
  Vocabulaire : Type par défaut, sous-type et paramètre de sous-type
  Fonction d'interrogation `KIND`
  Fonctions `SELECTED_INT_KIND` et `SELECTED_REAL_KIND`
  Spécificateur `KIND=` pour les fonctions intrinsèques

- Les fonctions qui renvoient les constantes machines de base :
  `HUGE , TINY , EPSILON , PRECISION , RANGE`
  `DIGITS , MAXEXPONENT , MINEXPONENT , RADIX`

- Les fonctions qui manipulent les nombres :
  `EXPONENT , FRACTION , SCALE , SET_EXPONENT`
  `NEAREST , RRSPACING , SPACING`

À ne plus utiliser … (sauf exceptions…)

Dimensionnement des tableaux avec `*` ou, pire, avec `(1)`

Déclaration des réels par `REAL*n` (jamais normalisé !)
Déclaration des entiers par `INTEGER*n` (jamais normalisé !)

Déclaration des chaînes de caractères par `CHARACTER*n`

Instruction `PAUSE` : à remplacer par `Read(*,*)`

Instructions `COMMON , EQUIVALENCE , BLOCK DATA , SEQUENCE`

Forme fixe du code source (col. 6, 7 et 72) pour du nouveau code !

Noms spécifiques pour les procédures intrinsèques (Fortran IV *i.e.* 66 !)

Fonction formule *ou* Fonction-Instruction (c'est traître à la lecture)

Descripteur d'édition `H` , `IF` arithmétique, Instruction `GOTO` calculé

IV : VERS UNE POO

Attribut `PUBLIC` et `PRIVATE` : On rentre dans la POO !

Définition de type dérivé : Instruction `TYPE` et `END TYPE`

```
Module PHYNU
   USE PRECISION
   Implicit None
   !
   Type NOYAU
      Integer :: A, Z, N
      Character(Len=2) :: El
   End Type NOYAU
   !
   Type NUCLIDE
      Type(NOYAU) :: Nuc
      Logical :: Stable = .True.    ! F95
      Real(SP) :: Lambda = 0._SP    ! F95
      Real(DP) :: Y = 0._DP         ! Abond. Mol.
   End Type NUCLIDE
   !
End Module PHYNU

Program TEST
   USE PHYNU
   Implicit None
   !
   Type (NOYAU) :: My_Nuc, Alpha
   Type(NUCLIDE) :: X, Y, Z
   !
   Alpha = NOYAU(4,2,2,"He")
   !
   My_Nuc = NOYAU(176,71,176-71,"  ")
   My_Nuc%El = SYMBOL(My_Nuc%Z)          ! i.e. Lu
   !
   X%Nuc = NOYAU(12,6,6," C")            ! i.e. C12
   !
   Y = Alpha .REACT. X
   Z = .REAC_AG. X
                          ! i.e. Y = Z = O16
   !
End Program TEST
```

Définition (déclaration), Instanciation et Initialisation :

Référence (accès) à un champ : %

Initialisation complète des objets dérivés en F95 (constructeur, destructeurs en f2k)

Création d'opérateurs et Surcharge d’opérateurs préexistants :
Voir exemple précédent …

Obsolescence (Définie dans la Norme) :

- Détruits en F95 :
  1. Boucles DO avec des réels
  2. Branchement à une instruction `END IF`
  3. Instruction `PAUSE` : à remplacer par `Read(*,*)`
  4. Instruction GOTO assignée et Formats assignés
  5. Descripteur d'édition `H`

- Prévu pour être détruit :
  1. `IF` arithmétique
  2. Fins d'instructions DO multiples et/ou partagées
  3. Instruction `RETURN` multiple
  4. Instruction `GOTO` calculé
  5. Fonction formule *ou* Fonction-Instruction
  6. Instruction `DATA` parmi les instructions executables
  7. Fonction caractère dont le résultat est de longueur `*`
  8. Forme fixe du code source (col. 6, 7 et 72)
  9. Déclaration des chaînes de caractères par `CHARACTER*n`
  10. Dimensionnement des tableaux avec `*` (??)

Nouveautés en cours de Fortran 2000 :

1. Traitement des exceptions IEEE : Bloc `ENABLE , END ENABLE`
2. Interopérabilité entre le C et le Fortran
3. Format de sortie `DT` pour les types dérivés
4. Héritage et autres paradigmes pour un L.O.O.
5. ... et bien d'autres choses !

Instructions pouvant devenir obsolétes (dépréciées) :

Instructions `SEQUENCE , EQUIVALENCE`
Instructions `COMMON , BLOCK DATA`
Directive `INCLUDE` (??)
Instruction `DO WHILE`
Écriture des `DOUBLE PRECISION`
Noms spécifiques pour les procédures intrinséques