FORTRAN en bref pour la programmation numérique

Patrick Amestoy, Philippe Berger, Ronan Guivarch,
François-Henry Rouet

PLAN

- Introduction
- Éléments du langage
- Instructions usuelles
- Programme et sous-programmes
- Utilisation des pointeurs
- Autres instructions
- Modules
- Format F66/77
- Fonctions intrinsèques
- Entrées/Sorties

Fortran

- Fortran (FORmula TRANslator) existe depuis les années 50 et a subi de nombreuses évolutions (dernière norme : Fortran 2008).
- Utilisé dans d'innombrables applications scientifiques :
  - Code Aster (EDF) : thermo-mécanique (>1M lignes de code).
  - NASTRAN (NASA & MSC Software) : thermo-mécanique, mécanique des fluides, acoustique (>1M lignes de code).
  - Mercator Océan : prévisions océanographiques.
  - Bloomberg : calculs financiers (25M lignes de code).
  - ...

Réferences

- "**Fortran 95/2003 Explained**", Michael Metcalf, John Reid , Malcolm Cohen. Oxford University Press, Aug 26, 2004
- "**The Fortran 2003 Handbook: The Complete Syntax, Features and Procedures**", J.C. Adams, W.S. Brainerd, R.A. Hendrickson, R.E. Maine, J.T. Martin, B.T. Smith. Springer

Fortran à l'ENSEEIHT

- 1ère année : « Calcul Scientifique » : Fortran utilisé pour de l'algèbre linéaire numérique.
- 2ème année : « Algèbre Linéaire Creuse », parcours HPC et Big Data
- 3ième année : « Calcul Répartie et Grid Computing » dans les parcours HPC et Big Data, Systèmes Logiciels et Infrastructure du Big Data pour l'IOT ainsi que le master recherche

Alphabet, identificateurs

Alphabet ::= alpha-numérique | caractère-spécial

alpha-numérique::= lettre | chiffre

lettre ::= majuscule | minuscule

majuscule ::= A | B | ... | Z

minuscule ::= a | b | ... | z

chiffre ::= 0 | 1 | ... | 9

c**aractère-spécia**l ::= * | / | + | - | < | > | < = | > = | = | /=

| _ | | (|) | " | ' | # | & | : | ;

Ident ::= lettre suivie d'un nombre quelconques de caractères alpha-numériques

Minuscule = Majuscule (ex : titi = TiTi)

Mots réservés, commentaires

Mot-clé ::= automatic, block data, byte, character, common, complex, data, dimension, double precision, equivalence, external, implicit, integer, intrinsic, logical, map, namelist, parameter, real, record, structure, union, program, function, subroutine, call, return, backspace, close, endfile, inquire, locking, open, read, rewind, write, case, case default, else, end if, end select, if, select case, continue, cycle, do, do while, end do, exit, allocate, deallocate

Commentaire ::= ! ceci est un commentaire

Types, variables, constantes

Langage ADA

Les types :
integer
boolean
character
float

Les déclarations :

```
x : integer ;
z : float ;
u, t : integer ;

pi : constant float := 3.1416 ;
```

Langage Fortran

Les types :
integer
logical (booléen)
character
real (simple précision ieee)
double precision
complex (2 réels simple précision)

Les déclarations :

```
integer x                              (F66/77)

double precision :: z                  (F90)
integer :: u, t                        (F90)

real, parameter :: pi = 3.1416  (F90)
```

Littéraux

- Constante entière (standard 32 bits) $-2147483648 \leq n \leq +2147483647$
- Constante réelle simple précision (32 bits IEEE) :

$$1.18\ 10^{-38} \leq |x| \leq 1.79\ 10^{+38}$$

  - Sous format virgule fixe : **3.1416**
  - Sous format virgule flottante : **0.31416E+1**
- Constante réelle double précision (64 bits IEEE) :

$$2.23\ 10^{-308} \leq |x| \leq 1.79\ 10^{+308}$$

  - Sous format flottant uniquement : **0.31416D+1**
- Constante texte
  - Constante caractère : **'Y'**
  - Constante chaîne de car. : **'BIEN L''BONJOUR' ou "Bonjour"**
- Constante logique : **.true. .false.**

Tableau statique

- Seules structures F66 / 77 : dimensions connues à la compilation (au plus 7)
- Rangement : tableau une dimension avec indices variant de gauche à droite
  - **matrice rangée colonne par colonne**
  - accès plus rapide par colonne (incrémentation adresse) que par ligne (calcul d'adresse avec multiplication impliquant le nb. de lignes)
- Déclaration

  real A(20),B(100,100),C(100,100) (F66/77)

  real, dimension(20) :: A (F 90)

  real, dimension(100,100) :: B,C

- 1er indice = 1, ou avec contraintes

Tableau Dynamique

En F90, variables tabulées dynamiques :

- déclaration (nb. de dimensions mais sans leur valeur)

  real, allocatable, dimension(:,:) :: A
- instruction de réservation mémoire (au cours de l'exécution)

  allocate(A(n,m))

  OU syntaxe étendue : **allocate(A(n,m), stat = retour)**

  retour type entier : **retour** = 0 allocation réussie, **retour** > 0 échec
- Après utilisation, mémoire libérée par instruction :

  deallocate(A)

Manipulation de tableau

- indice de 1 à la taille du tableau, sauf si spécifié au moment de la déclaration :

  integer, dimension(-5 :10) :: E

- pas de vérification que l'indice est dans le bon intervalle (indication à la compilation avec certains compilateurs quand c'est possible)
- accès à un élément **:**

  v(i), m(i,j), ...

Pointeurs

Une variable de type pointeur (F90 uniquement) se déclare comme associé à un type d'objet :

integer, pointer :: p

integer, target :: n

p un pointeur sur variables entières, **n** entier susceptible d'être « pointé » (attribut **target**)

Enregistrements

Déclaration type non standard en F90 uniquement

- déclaration du type : type point : 2 champs de type réel

```
type point
   real :: abscisse
   real :: ordonnee
end type point
```

- déclaration de variables : x, y, z : 3 variables de type point

```
type (point) :: x, y, z
```

- accès aux champs

x%abscisse pour accès au champ abscisse de **x**

y%ordonnee pour accès au champ ordonnee de **y**

Les Expressions

- **Opérateurs Arithmétiques**
  - Par priorité décroissante :

    (1) ** puissance

    (2) * et /

    (3) + et -
  - minimum de parenthèsage :

    $ax^2 + bx + c$ → **a*x**2+b*x+c**
  - Expression mixte (avec ≠ types) :

    type résultat : → type le plus complexe : (réel + entier) → réel
  - Conversion automatique source d'erreurs :

    → emploi de fonctions intrinsèques de conversion

Expressions

- **Opérateurs Arithmétiques** : application
  - aux objets élémentaires,
  - aux variables tabulées prises dans leur globalité :

    real, dimension(2,2) :: a, b, e

    e = a + b
  - et aux sections conformes de variables tabulées :

    real, dimension(2,2) :: a

    real, dimension(4,4) :: c, d

    a = c(1:2, 1:2) + d(1:3:2, 1:3:2)
  - Section (d'une matrice **d**) : **d(i_1 :i_2 :i_3, j_1 :j_2 :j_3)** ≡ lignes de **i_1** à **i_2** avec pas **i_3** (optionnel), colonnes de **j_1** à **j_2** avec pas **j_3** (optionnel)
  - Sections conformes : même nb. de dim., même nb. d'éléments par dim

    a(1,1)=c(1,1)+d(1,1); a(1,2)=c(1,2)+d(1,3) ;

    a(2,1)= c(2,1)+d(3,1); a(2,2) = c(2,2) + d(3,3)

Expressions

- **Opérateurs relationnels** : applicables aux objets élémentaires, variables tabulées, sections conformes de variables tabulées

Syntaxe F77	Syntaxe F90	Syntaxe F77	Syntaxe F90
a.gt.b	a>b	a.lt.b	a<b
a.ge.b	a>=b	a.le.b	a<=b
a.eq.b	a==b	a.ne.b	a/=b

- **Opérateurs logiques** : sur booléens élémentaires, tableaux de booléens, sections conformes de tableaux de booléens
  - par priorité décroissante :

(1) **.not.** (non logique)

(2) **.and.** (et logique)

(3) **.or.** (ou logique)

(4) **.eqv.** (équivalence), **.neqv.** (non équiv.)

Instruction : séquence

- F66/77 : 1 instruction par ligne

  si 2 (ou +) lignes nécessaires : ligne suite formaté

- F90 : plusieurs instructions / ligne → séparateur **;**

  si 2 (ou +) lignes nécessaires, caractère **&** comme symbole annonçant la ligne suivante

Langage ADA

```
- - Séquence
X := 2 ; Y :=  4 ;
Z := a + c +
       d - e ;
```

L'affectation :=

Langage Fortran (F90)

```
! Séquence
X = 2 ; Y = 4
Z = a + c + &
      d - e
```

L'affectation =
conversion automatique peu fiable

Instruction : Si Alors Sinon

[nom :] **if** (<*exp logique*>) **then**

<*bloc d'instr1*>

elseif (<*exp logique*>) **then**

<*bloc d'instr2*>

else

<*bloc d'instr3*>

end if [nom]

Langage ADA

```
 - -   x a une valeur
if (x >= 0) then
    y := x ;
else
   y := -x ;
end if ;
 - -  y = |x|
```

Langage Fortran

```
! x a une valeur
if (x >= 0) then
   y = x
else
   y = -x
end if
! y = |x|
```

Autres formes du Si Alors Sinon

- test logique simple

  if (<*exp logique*>) <*inst simple*>
- forme simplifiée sans sinon

  if (<*exp logique*>) **then**

    <*bloc d'inst*>

  end if
- **elseif** existe

Instruction : Cas de

Langage ADA

```
case x is

  when 0 => ...
  when 1|4 =>  ...
  when  5, 7 => ...
  when others => …

end case ;
```

Langage FORTRAN

```
select case (x)
   case (0)
      ...
   case (1 : 4)
      ...
   case (5, 7)
      …
   case default
      ...
end select
```

Boucle Bornée Pour

[nom :] **do** <*ind*> **=** <*binf*>, <*bsup*> *[, <pas>]*

<*bloc instr*>

end do [nom]

<pas> : incrément (1 si omis, possibilité de pas négatif)

Langage ADA

```
- - précondition
- - N et x ont une valeur

Res := 1 ;

for i in 1 .. N  loop
  Res := Res * x ;
end loop ;

- - Res contient x ** n
```

Langage Fortran

```
! précondition :
! N et x ont une valeur

Res = 1

b1 : do i = 1, N
    Res = Res * x
end do b1

! Res contient x ** n
```

Boucle non bornée Tant Que

[nom :] **do while** (<*expr logique*>)

<*bloc d'instr*>

end do [nom]

Langage ADA

```
x := 1 ;

while  (x< n)  loop

   x := x * 2 ;

end loop ;

- - x vaut la première puissance
- - de 2 supérieure à N
```

Langage Fortran

```
x = 1

do while  (x <  n)

    x = x * 2

end do

! x vaut la première puissance
! de 2 supérieure à N
```

Entrées/Sorties simples

En sortie : **write(*,*)** *<liste de variables>*

En entrée : **read(*,*)** *<liste de variables>*

(*,*) : (périphériques par défaut (écran | clavier), format par défaut)

En sortie : **print ***, *<liste de variables>*

En entrée : **read ***, *<liste de variables>*

Langage ADA

compliqué !

Langage Fortran

```
read(*,*) v
read(*,*) v1, v2, ..., vn

write(*,*) 'indice = ', i, 'valeur =', v
print *, 'indice = ', i, 'valeur =', v
```

Forme d'un programme

Langage algo

```
procedure SommeCarres is

  déclarations de constantes
  déclarations de types
  déclarations de variables

begin

   instructions

end SommeCarres ;
```

Langage Fortran

```
program SommeCarres

  ! déclarations de constantes
  ! déclarations de types
  ! déclarations de variables

  ! instructions

end program SommeCarres
```

Exemple de Programme : calcul du nombre d'or
Exercice : le tri bulle

Procédure

• définition

subroutine <*nom proc*>(<*p1*>,...,<*pn*>)
    <*déclaration type des paramètres*>
    <*décl. constantes, types, variables locales*>
    <*liste instructions*>
end subroutine <*nom proc*>

• appel
  call <*nom proc*>(<v*p1*>,...,<v*pn*>)

• mode des paramètre au niveau de la déclaration des types des paramètres avec le mot clé **intent** : **in, inout, out**

• dernière instruction exécutée : celle qui précède le **end subroutine**

Un exemple de procédure

```
! procédure init_tab
! initialise un vecteur de réels
! t : le vecteur (résultat)
! n : sa dimension (donnée)
! post-condition : t(1:n) initialisés

subroutine init_tab(t, n)
  implicit none
  integer, intent(in) :: n
  real, intent(out), dimension(n) :: t
  integer :: i

  print *, "rentrez ", n, "réels"
  do i = 1, n
    print *, "indice ", i
    read(*,*) t(i)
  end do
  ! les dim valeurs sont initialisées
  return
end subroutine init_tab
```

```
! programme de test
program test
  implicit none
  integer :: n
  parameter(n=10)
  real, dimension(n) :: v, w
  integer :: taille

  ! choix de la taille réelle
   ...
  ! 0 < taille <= n

  ! initialisation des 2 vecteurs
  call init_tab(v, taille)
  call init_tab(w, taille)

 ! affichage des 2 vecteurs
 ! (taille réelle)
  print *, "v = ", v(1:taille)
  print *, "w = ", w(1:taille)

 end program test
```

Exercice : transformer le tri bulle en procédure

Fonctions

- définition

<type résultat> function <*nom fonction*>(<*p1*>,….,<*pn*>)

<*décl. type paramètres*>

<*décl. constantes, types, variables locales*>

<*liste instructions*>

end function <*nom fonction*>

- résultat :
  - l'identificateur <*nom fonction*> permet de manipuler le résultat comme une variable
  - <*liste instructions*> comporte au moins une affectation de <*nom fonction*>
- appel
  - dans l'appelant, le type de la fonction doit être défini dans la partie déclarative
  - syntaxe de l'appel : <*nom fonction*>(<v*p1*>,….,<v*pn*>)

Un exemple de fonction

```
! fonction ps
! calcule le produit scalaire de 2 vecteurs
! v1, v2 : les 2 vecteurs (données)
! n : leur dimension commune (donnée)
! pré-conditions :
!    v1(1:n) et v2(1:n) initialisés
real function ps  (v1, v2, n)
  implicit none
  integer, intent(in) :: n
  real, intent(in), dimension(n) :: v1, v2
  integer :: i

  ! initialisation
  ps = 0.0
  ! calcul de v1.v2
  do i=1,n
    ps = ps + v1(i) * v2(i)
  end do
  ! ps contient v1(1:n).v2(1:n)
  return
end function ps
```

```
! test de la fonction
program test
  implicit none
  ! variable du programme
  integer :: n
  parameter(n=10)
  real, dimension(n) :: v, w
  integer :: taille

  ! fonction(s) utilisée(s)
  real :: ps

  ! choix de la taille réelle
   ...

  ! initialisation des 2 vecteurs
  ...

  ! affichage du produit scalaire
  print *, ps(v, w, taille)

end program test
```

Exercices :

- schéma de Hörner pour calculer la valeur d'un polynôme en un point (avec allocation dynamique)

Importance de la dimension principale d'un tableau :

- Exemple simple : initialisation d'une matrice
- Factorisation de Gauss

Fin TD1

Pointeurs

- Mise en relation avec variable statique déjà en mémoire **=>**:

  integer, target :: n

  integer, pointer :: p

  n = 5 ; p **=>** n ! p pointe sur l'emplacement contenant n (= 5)

- Création dynamique de la variable pointée : **allocate**

  integer, pointer :: p

  allocate(p, stat=retour)

  réservation d'une zone pour un entier,

  p contient l'adresse, **retour** le diagnostic de l'allocation

- Fonction **associated**(p) retourne **.true.** si **p** associé **.false.** sinon
- Instruction **nullify**(p) :
  - affecte 0 au pointeur **p** (**p** n'est plus associé)
  - toute tentative d'accès à la zone pointée par **p** => échec du programme
- Libération de la zone créé dynamiquement : **deallocate**

  deallocate(p) ! zone libérée et p n'est plus associé

Pointeurs

- Affectation **=** modification de la valeur de l'objet pointé

```
integer, target :: n,m
integer, pointer :: p, q
n = 5 ;  m = 98 ;  p => n ; q => m
m = m + p    ! m prend la valeur 103
p = 45       ! comme p pointe sur n, n prend la valeur 45
p = q        ! l'objet pointé par p reçoit la valeur de l'objet pointé par q (n reçoit 103)
```

- Affectation **=>**

```
integer, target :: n, m
integer, pointer :: p, q
n = 5 ;  m = 98
p => n       ! p pointe sur n
q => m       ! q pointe sur m
p => q       ! p et q pointent sur m
```

Pointeurs

- Si **p** et **q** pointeurs, il ne faut pas confondre :
  - **p => q** : **p** et **q** pointent sur le même objet (celui initialement pointé par **q**)
  - **p = q** : l'objet pointé par **p** reçoit la valeur de l'objet pointé par **q**
- pointeur associé à une structure (enregistrement) :

```
type point
  real :: abscisse
  real :: ordonnee
end type point
type (point), pointer :: p
allocate(p, stat=retour) ! crée un objet de type point, pointé par p
p%abscisse = 3.0          ! pour accéder à un des champs de la structure
```

Autres instructions FORTRAN

- que vous pourrez rencontrer dans des codes existants
  - Arrêt d'un programme **stop**
  - Branchement inconditionnel **goto** <*étiquette*>

    (F77 pour écrire des boucles non bornées)
  - Suite en séquence **continue**
  - sortie de boucle **exit**

Exemple de **goto** pour traiter les cas d'erreur

```
program euclid
  implicit none
  integer :: a, b  ! 2 entiers dont on cherche le pgcd
  integer :: erreur  ! gestion des erreurs de saisie
  integer :: pgcd    ! fonction pgcd

  print *, 'A?'
  read(*,*) a
  if (a <= 0) then
    erreur = 1
    goto 100
  end if
  print *, 'B?'
  read(*,*) b
  if (b <= 0) then
    erreur = 2
    goto 100
  end if
  print *, 'le pgcd de ', a, ' et', b, ' est', pgcd(a, b), '.'
  stop
```

```
    ! traitement des erreurs
100 continue
    select case (erreur)
      case(1)
        print *, 'a doit être strictement positif.'
      case(2)
        print *, 'b doit être strictement positif.'
      case default
        print *, 'on ne devrait jamais afficher cela'
    end select

  end program euclid
```

Modules

- rôle : partage de données, de types, de traitements entre plusieurs unités de compilation
- définition

  module <*nom module*>

  <*partie déclarative*>

  contains

  <*suite de procédures et/ou fonctions*>

  end module <*nom module*>

- utilisation : l'unité qui fait référence à un module comporte en en-tête la directive :

  use <nom module>

Un module associé à un objet

- Le module **mat_tridiag** comprend :
  - la définition du type tridiag (matrice tridiagonale)
  - tous les traitements manipulant ce type d'objet (par ex., somme et soustraction de 2 matrices)

```
module mat_tridiag
  type tridiag
    real,dimension(10):: diag,diagsup,diaginf
  end type tridiag
contains
        subroutine somme (a, b, c)
          type (tridiag), intent(in) :: a, b
          type (tridiag), intent(out):: c
          c%diag = a%diag + b%diag
          c%diagsup = a%diagsup + b%diagsup
          c%diaginf = a%diaginf + b%diaginf
        end subroutine somme
!
        subroutine soustraction (a, b, c)
          type (tridiag), intent(in) :: a, b
          type (tridiag), intent(out):: c
          c%diag = a%diag - b%diag
          c%diagsup = a%diagsup - b%diagsup
          c%diaginf = a%diaginf - b%diaginf
        end subroutine soustraction
end module mat_tridiag
```

```
program validation
  use mat_tridiag
!
  type (tridiag) :: m1, m2, m3
!
! Initialisation de m1 et de m2
  call somme(m1, m2, m3)
!
end program validation
```

Un module définissant des sous-programmes génériques

- Ici module **mat_tridiag** comprend juste la définition du type tridiag
- **addition_matrice** :
  - comprend les procédures effectuant la somme de 2 matrices pour différentes topologies
  - ces procédures sont référées sous le même nom somme par une **interface**

```
program validation
  use mat_tridiag
  use addition_matrice
  type (tridiag) :: m1, m2, m3
  real, dimension(10,10) :: n1, n2, n3
  ! on suppose ces 6 matrices déjà initialisées
  call somme(m1, m2, m3)
  call somme(n1, n2, n3)
end program validation

module mat_tridiag
  type tridiag
    real,dimension(10):: diag,diagsup,diaginf
  end type tridiag
end module mat_tridiag
```

```
module addition_matrice
! ce module doit connaître le type tridiagonale
  use mat_tridiag
! « interface » qui assure la généricité
  interface somme
    module procedure s_pleine, s_tridiag
  end interface somme
contains
  subroutine s_tridiag (a, b, c)
    ! code : voir transpa précédent
   end subroutine s_tridiag
!
  subroutine s_pleine (a, b, c)
    ! somme de 2 matrices pleines : c = a + b
    real, dimension(10,10), intent(in) ::a, b
    real, dimension(10,10), intent(out):: c
     c = a + b
  end subroutine s_pleine
end module addition_matrice
```

Format Fortran 66 et 77

Unité de compilation (programme, procédure, fonction) ≡ lignes formatées de 72 caractères à 3 champs :

- Col 1 - 5 : zone étiquette

  Etiquette ≡ nombre d'au plus 5 chiffres qui identifie l'instruction qui suit sur la même ligne. Généralement utilisée avec une instruction de saut inconditionnel :

  goto <étiquette>

  L'exécution se poursuit par l'instruction marquée par l'étiquette

- Col 6 : zone suite

  Si caractère autre que ' ' et '0' ≡ ligne suite (utilisée lorsqu'une instruction ne peut être codée sur une seule ligne)

- Col 7 - 72 : zone texte

  Déclarations de données, instructions (une seule instruction par ligne)

Fonctions Intrinsèques

- F66 / 77 : nombreuses fonctions intrinsèques (incluse dans la norme du langage) essentiellement arithmétiques
- F90 : ajout d'opérateurs plus orientés vers manipulation de données
- Exemples de fonctions intrinsèques arithmétiques (F66 / 77)
  - En simple précision (opérande et résultat) : **cos**(x) (cosinus), **exp**(x) (exponentielle), **abs**(x) (valeur absolue), …
  - En double précision, en général ajout de 'D' devant le nom de la fonction : DCOS, DEXP, DABS, …
- Exemples de fonctions intrinsèques de conversion (F66 / 77)
  - DFLOAT(2.4) convertit 2.4, initialement simple précision, en double précision
  - INT(2.4) prend la partie entière de 2.4 (renvoie donc la valeur entière 2)

Fonctions Intrinsèques

- F90, fonctions arithmétiques ou de conversion peuvent s'appliquer à des opérandes type section de tableau : **exp**(a(4 :9)) génère un tableau de 6 éléments (exponentielle des coefficients de a de l'indice 4 à 9)
- Exemples de nouvelles fonctions intrinsèques incluses dans F90
  - **epsilon**(x) : précision machine associée à une variable du même type que x
  - **ishift**(i,dec) décale l'entier i de dec bits vers la gauche (nouveaux bits =0)
  - **dot_product**(a,b) : produit scalaire de 2 sections de vecteur conformes a et b
  - **matmul**(a,b) : produit matriciel de 2 sections de matrice a et b
  - **maxval**(a) : plus grande valeur de la section a
  - **sum**(a) : somme des éléments de la section a
  - **size**(a,dim) : nombre d'éléments de la section a suivant la dimension dim

Fonctions Intrinsèques (bonus)

- **pack(**a,masque**)** (compression)
  - **a** section de vecteur,
  - **masque** section d'un vecteur de booléens (même nb d'éléments que **a**)
  - Résultat : vecteur composé des éléments de **a** pour lesquels le booléen associé dans **masque** est vrai
  - exemple : **a** : [1 2 3 4 5]

    masque : [.false. .false. .true. .false. .true.]

    résultat : [3 5]
- **merge**(a, b, masque) (fusion)
  - a et b sections conformes composées d'éléments de même type, masque section d'un vecteur de booléens (même nb d'éléments que a ou b)
  - Résultat : section conforme à a ou b construite à partir d'éléments de a pour les indices où masque est vrai, de b pour les indices où masque est faux
  - exemple : **merge** (a, b, a>b)

    a : [1 2 3 4 5 6 7 8 9]

    b : [0 0 1 8 1 9 8 2 1]

    résultat : [1 2 3 8 5 9 8 8 9]

Entrées/Sorties non standard

- de manière générale
  - sortie : **write**(<unité>,<format>)<liste variables>
  - entrée : **read**(<unité>,<format>)<liste variables>

  avec
  - <unité> : numéro logique associé à un périphérique physique (clavier (5), écran (6), imprimante, …) ou logique (fichier).

    Par défaut, caractère '*' (en entrée : clavier, sortie : écran)
  - <format> : étiquette référençant un format sous lequel s'effectue l'E/S

```
        write(6,100)i,A(i)
100     format('INDICE = ',I4,' VAL = ',E12.6)
        ! Variable entière i sur 4 caractères
        ! Variable réelle A(i) sur 12 caractères dont 6 décimales
```

  - à l'écran (6) : INDICE = ##34 VAL = -.112425E+23

Entrées/Sorties non standard

- Principaux formats

  In entier

  Ew.d réel simple précision

  Dw.d réel double précision

  Fw.d réel virgula fixe

  Il existe des caractères spéciaux pour affiner un format : **‘/’** saut à la ligne suivante, **‘nX’** saut de **n** caractères sur une même ligne ….

- Entrées/Sorties sur fichier avec un exemple : construction et mise à jour d’un fichier séquentiel de nombres réels avec formatage par défaut :

```
open(10, file='vecteur.txt')
do i = 1, 100
  write(10,*) a(i)
end do
close(10)
```