



Fortran 90 et 95

Dominique COLOMBANI - Web Formation

Table des matières

| | |
|--|-----------|
| 1. Introduction au langage Fortran | 5 |
| 2. Organisation et structure | 6 |
| 2.1. Organisation..... | 6 |
| 2.2. Structure d'une unité de compilation..... | 6 |
| 2.3. Format d'une ligne | 6 |
| 2.4. Jeu de caractères | 7 |
| 2.5. Présentation des modules | 7 |
| 3. Variables | 9 |
| 3.1. Présentation des variables..... | 9 |
| 3.2. Typage implicite | 9 |
| 3.3. Types de base..... | 10 |
| 3.3.1. INTEGER..... | 10 |
| 3.3.2. REAL..... | 10 |
| 3.3.3. KIND..... | 10 |
| 3.3.4. DOUBLE PRECISION | 12 |
| 3.3.5. COMPLEX..... | 12 |
| 3.3.6. LOGICAL..... | 13 |
| 3.3.7. CHARACTER..... | 13 |
| 3.4. Initialisation..... | 14 |
| 3.5. Portée et durée de vie..... | 15 |
| 3.6. Constantes..... | 16 |
| 3.7. Synonymes..... | 16 |
| 4. Expressions et opérateurs | 17 |
| 4.1. Expressions..... | 17 |
| 4.2. Opérateurs | 17 |
| 4.2.1. affectation..... | 17 |
| 4.2.2. arithmétiques..... | 17 |
| 4.2.3. relationnels..... | 18 |
| 4.2.4. logiques..... | 18 |
| 4.2.5. caractères..... | 19 |
| 4.2.6. Priorités..... | 19 |
| 4.3. Fonctions intrinsèques..... | 19 |
| 5. Définition de types - Fortran 95 | 22 |
| 5.1. Types - Fortran 95..... | 22 |
| 5.2. Initialisation - Fortran 95..... | 23 |

| | |
|--|-----------|
| 6. Structures de contrôle | 26 |
| 6.1. Tests..... | 26 |
| 6.2. Boucles..... | 27 |
| 6.3. Branchement inconditionnel : GOTO | 28 |
| 6.4. Rupture de boucles en Fortran 90 | 29 |
| 7. Fonction et sous programme | 31 |
| 7.1. Procédures..... | 31 |
| 7.1.1. Fonctions..... | 31 |
| 7.1.2. sous programmes..... | 32 |
| 7.1.3. Passage de paramètres | 34 |
| 7.1.4. Durée de vie..... | 37 |
| 7.1.5. Compléments sur les sous programmes et les fonctions..... | 37 |
| 7.2. Procédures - éléments avancés | 38 |
| 7.2.1. Emplacement de définition | 38 |
| 7.2.2. Interface | 38 |
| 7.2.3. Interface générique..... | 40 |
| 7.2.4. Fonctions récursives | 42 |
| 8. Tableaux | 44 |
| 8.1. Tableaux statiques | 44 |
| 8.2. Initialisation de coefficients | 44 |
| 8.3. Initialisation d'un tableau | 45 |
| 8.3.1. Instruction DATA | 45 |
| 8.3.2. Constructeur de tableau..... | 46 |
| 8.4. Stockage en mémoire | 47 |
| 8.5. Modifications de forme..... | 48 |
| 8.6. Adressage avancé | 50 |
| 8.7. Opération sur les tableaux - Fortran 90..... | 50 |
| 8.8. Tableaux dynamiques - Fortran 90..... | 56 |
| 9. Pointeur | 57 |
| 10. Surcharge d'opérateurs | 61 |
| 10.1. surcharge opérateur..... | 61 |
| 10.2. Surcharge operateur assignation..... | 62 |
| 11. Contrôle de visibilité | 63 |
| 11.1. Visibilité des champs - Fortran 95..... | 63 |
| 12. Entrée/Sortie | 65 |
| 12.1. Accès à un fichier | 65 |
| 12.2. Écriture et lecture..... | 66 |
| 12.3. Entrée/Sortie standard..... | 69 |

| | |
|---|-----------|
| 12.4. Interactions avec l'utilisateur | 69 |
| 12.5. Descripteurs de format | 69 |
| 12.6. Échange de données avec d'autres applications | 72 |
| 12.7. Déplacements dans un fichier | 72 |
| 12.8. Entrées Sorties particulières..... | 72 |
| 13. Fonctions intrinsèques | 74 |
| Index | 75 |
| Mentions légales | 77 |

3. Variables

3.1. Présentation des variables

Variable

Az Définition

- zone mémoire, identifiée par un nom
 - au maximum 31 caractères : lettres, chiffres et _
 - le premier caractère doit être une lettre
 - pas de différences entre majuscule et minuscule
- une variable a un type, défini une fois pour toute
 - spécifie ce qui peut être représenté (entier, caractère, réel, ...)
 - la façon dont est faite la représentation
 - les bornes des valeurs qui peuvent être représentées

Déclaration du type

- la déclaration doit être effectuée au début de l'unité de compilation
- la déclaration explicite n'est pas obligatoire, mais est fortement recommandé, pour éviter le typage implicite
- formes de la déclaration
 - type identifiant1, identifiant2 ! Fortran 77
 - type [,attribut1, attribut2] :: identifiant1, identifiant2 ! Fortran 90, avec des attributs optionnels

3.2. Typage implicite

- Les variables non déclarées ont un type implicite.
- Par défaut, la première lettre de l'identificateur définit le type
 - I, J, K, L, M, N : entier (I-N, début de INTEGER)
 - autre : réel
 - Très dangereux ...
- Instruction IMPLICIT permet de définir d'autres typages par défaut
 - IMPLICIT DOUBLE PRECISION (A-D)
 - Très dangereux également

² <https://onlinegdb.com/C3ZILWY-S>

Bonne pratique

- utiliser IMPLICIT NONE (ou l'option équivalente du compilateur) pour imposer la déclaration de chaque variable
- Options du compilateur
gfortran : -fimplicit-none
Intel : /warn:declarations (Windows) ou -warn declarations (Linux)

3.3. Types de base

3.3.1. INTEGER

- entier dont la taille dépend du compilateur
- constantes
 - décimale : 23
 - binaire : B'1101001111' (avec DATA)
 - octale : O'735' (avec DATA)
 - hexadécimale : Z'FA3' ou X'FA3' (avec DATA)
- spécification de taille
Fortran 77 : INTEGER*2, INTEGER*4
- Fortran 90 : INTEGER(KIND=2), INTEGER(KIND=4) ou INTEGER(2), INTEGER(4)

3.3.2. REAL

- valeurs numériques sur 4 octets
- $1.2 \cdot 10^{-38} \leq |r| \leq 3.4 \cdot 10^{38}$ avec 6 chiffres significatifs.
- constantes :
0.123
.123
1.23E-1
-0.123
- spécification de taille
Fortran 77 : REAL*4 (identique à REAL), REAL*8 (DOUBLE PRECISION)
Fortran 90 : REAL(KIND=4), REAL(KIND=8) ou REAL(4), REAL(8)

3.3.3. KIND

Détermination d'une valeur de KIND

- SELECTED_INT_KIND(m) : valeur de KIND pour les nombres compris entre -10^m et 10^m
renvoie -1 si impossible
- SELECTED_REAL_KIND((p,r) : valeur de KIND pour les nombres tels que $10^{-r} < |x| < 10^r$ avec une précision p

renvoie

-1 si la précision demandée n'est pas disponible

-2 si l'étendue n'est pas disponible ;

-3 si la précision et l'étendue ne sont pas disponibles.

```

1 program select_kind
2 implicit none
3
4 print*,selected_int_kind(8)      ! 4
5 print*,selected_int_kind(2)     ! 1
6 print*,selected_int_kind(20)   ! 16
7 print*,selected_int_kind(40)   ! -1
8 print*, 'Flottant -----'
9 print*,selected_real_kind(8,5)  ! 8
10 print*,selected_real_kind(4,5) ! 4
11 print*,selected_real_kind(4,100) ! 8
12 print*,selected_real_kind(8,400) ! 10
13 print*,selected_real_kind(8,5000) ! -2
14
15 end program select_kind

```

Exécuter ce code³

valeur de kind

Remarque

- la norme n'impose pas que les valeurs de kind soient la taille en octets (4 ou 8). Il peut exister des compilateurs avec des valeurs 1, 2, ...
(=> documentation du compilateur)
- on peut utiliser un paramètre pour kind
integer, parameter :: r=4
real(kind=r) :: x
- on peut définir une constante avec le type associé à kind avec la notation `_kind`
x = 3.0_r ; y=5.0_4

```

1 program demo_kind
2 implicit none
3 integer, parameter :: r=4
4 real(kind=r) :: x,y
5 x = 3.0_r ; y=5.0_4
6 print*,x,y
7
8 end program demo_kind

```

Exécuter ce code⁴

RANGE, PRECISION

RANGE fournit la plage de valeur

PRECISION fournit le nombre de chiffres significatifs

³ <https://onlinegdb.com/d0ZGxV76qw>

```

1 program demo_range_precision
2 implicit none
3 real :: r
4 real(8) :: d
5 integer :: i
6 integer(2) :: j
7 print*, range(r), precision(r)
8 print*, range(d), precision(d)
9 print*, range(i)
10 print*, range(j)
11 end program demo_range_precision

```

Exécuter ce code⁵

3.3.4. DOUBLE PRECISION

- valeurs numériques sur 8 octets
- $2.2 \cdot 10^{-308} \leq |r| \leq 1.8 \cdot 10^{+308}$ avec 15 chiffres significatifs
- constantes :
 - 0.123D0
 - .123D0
 - 1.23D-1
 - 0.123D0
- peut être déclaré avec une instruction REAL*8 au lieu de DOUBLE PRECISION

```

1 program declar
2 implicit none
3 real*8 pi
4 pi = 3.14d0
5 write (6,*) pi
6 end program declar

```

Exécuter ce code

```

1 program precision
2 real pi /3.14/
3 real*8 double_pi /3.14D0/
4 real(kind=4) :: r
5 real(kind=8) :: r2
6 r = 50.
7 r2 = 5.D1
8 write(6,*) r * pi -157.
9 write(6,*) r2 * double_pi -157.D0
10 end program precision

```

Exécuter ce code

3.3.5. COMPLEX

- nombre complexe dont les parties réelle et imaginaire sont des REAL
- COMPLEX*16 pour des double précision
- constante :
 - (2.5, 1.) représente $2.5 + i$
- fonctions intrinsèques REAL et IMAG pour accéder aux parties réelles et imaginaires

4. https://onlinegdb.com/8h_cah0JM

```

1 Program demo_complexe
2 implicit none
3 complex :: c
4 c = (10.,1.)
5 print*,c ! (10.0000000,1.00000000)
6 print*, "Partie réelle ", real(c) ! 10.0000000
7 c = c * c
8 print*, c ! (99.0000000,20.0000000)
9 end

```

Exécuter ce code⁶

3.3.6. LOGICAL

- valeurs booléennes
- .TRUE. , .FALSE.

```

1 program prog
2 integer*4 i,j
3 logical ok, ko
4 ok = .true.
5 ko = .false.
6 write (6,*) ok, ko
7 i=1
8 j=2
9 write(6,'(1X,L10)') i .eq. j
10 end

```

Exécuter ce code

3.3.7. CHARACTER

- code ASCII
- définition d'une longueur
 - CHARACTER(LEN=n) nom
 - CHARACTER*n nom
 - CHARACTER nom*n
- constantes " " ou ' '
- (attention : chaîne complétée par des espaces lors d'une affectation)
- ancienne forme de constante, utilisée pour des initialisations ou des formats
constantes Hollerith : DATA TXT /7Hbonjour/
(nombre de caractères, lettre H et caractères)
- sous chaîne CH(5:7)
si on ne spécifie pas la borne, la valeur minimale ou maximale est utilisée
 - CH(:3) est équivalent à CH(1 3)
 - CH(3:) est équivalent à CH(3:LEN(CH))

```

1 program demo_caractere
2 character(len=10) texte1
3 character*5 texte2
4
5 texte1="Bonjour"
6 texte2='Au revoir'
7

```

⁶ <https://onlinegdb.com/1BZMDBkm->


```

8 print*,texte1      ! Bonjour
9 print*,texte2      ! Au re
10 print*,texte1(1:3) ! Bon
11 print*,texte1//texte2 ! Bonjour  Au re
12
13 end program demo_caractere

```

Exécuter ce code⁷

character(*)

 Remarque

On peut utiliser character(*) dans plusieurs constextes

- la chaine est une constante
character(*), parameter :: fort = "Fortran"
- la chaine est un argument d'une procédure

function f1(text)

character(*) text

```

1 program demo_caractere
2 character(*), parameter :: texte1 = "Bonjour"
3 character :: texte2*5 = 'Au revoir'
4 character(80) upper
5
6 print*,texte1      ! Bonjour
7 print*, len(texte1) ! 7
8 print*,texte2      ! Au re
9 print*,upper(texte1)
10
11 end program demo_caractere
12
13 character(80) function upper(t)
14 character(*),intent(in) :: t
15 integer :: i
16 character(len(t)) :: tm
17 do i = 1, len(t)
18     if (t(i:i) >= 'a' .and. t(i:i) <= 'z') then
19         tm(i:i) = achar(iachar(t(i:i))-32)
20     else
21         tm(i:i) =t(i:i)
22     end if
23 end do
24 upper = tm
25 end function upper

```

Exécuter ce code

3.4. Initialisation

Fortran 77

- On peut initialiser une variable avec l'instruction DATA
DATA liste_variables / liste_valeurs /
- les valeurs doivent correspondre aux types des variables

⁶ <https://onlinegdb.com/-PqlxQkpST>

- Une variable initialisée a souvent une durée de vie égale à celle du programme (attribut SAVE automatique) : cela dépend du compilateur
- L'initialisation peut être faite lors de la déclaration
REAL PI /3.14/

Fortran 90

- Initialisation REAL :: PI = 3.14 ! les :: sont obligatoires

3.5. Portée et durée de vie

Portée

- Une variable est définie dans une unité de compilation
- Elle n'est pas connue en dehors de cette unité

```

1 program portee
2 implicit none
3 real :: x
4 x = 3.
5 call ssprog1
6 end program portee
7
8 subroutine ssprog1
9 implicit none
10 print*,x ! Error: Symbol 'x' at (1) has no IMPLICIT type
11 end subroutine ssprog1

```

Exécuter ce code⁸

Durée de vie

- Par défaut, une variable (donc la valeur qu'elle a) est détruite à la fin de l'unité où elle est déclarée.
- Pour la rendre persistante, il faut utiliser SAVE
- Fortran 77
REAL X
SAVE X
- Fortran 90
REAL, SAVE :: X
- attention
aux initialisations qui font un SAVE implicite
aux options de compilation (-fautomatic, -fno_automatic de gfortran)

```

1 program duree_de_vie
2 call affiche
3 call autre
4 call affiche
5 end
6 subroutine affiche
7 real :: x = 3.
8 real :: y
9 print*, 'Valeur de x dans affiche : ',x
10 print*, 'Valeur de y dans affiche : ',y
11 x = 10

```

7. <https://onlinegdb.com/sqkPbJLVF>

```
12 y = 42
13 end
14 subroutine autre
15 integer :: i
16 i = 1000
17 print*, 'Valeur de i dans autre : ',i
18 end subroutine autre
```

Exécuter ce code⁹

NB : tester en commentant l'appel à autre

3.6. Constantes

Fortran 77

- On peut définir des constantes avec l'attribut PARAMETER
- REAL PI
PARAMETER (PI=3.14)
- on peut utiliser des expressions, par exemple 22/7 ou 4*ATAN(1.)

Fortran 90

REAL, PARAMETER :: PI = 3.14

3.7. Synonymes

- On peut définir des synonymes avec l'instruction EQUIVALENCE
- Il y a juste correspondance des emplacements de variables, sans conversion
- COMPLEX C
REAL R
EQUIVALENCE (R,C)

```
1 program equiv
2 complex :: c = (3.,2.)
3 real :: r
4 equivalence (r,c)
5 print *,r
6 print *,c
7 r = 0
8 print *,c
9 end
```

Exécuter ce code

⁸ <https://onlinegdb.com/TZfi3mMSh>