
Programmation impérative: Méthode de Programmation

5. Structures de données

Les structures, pour quoi faire ?

Objectifs :

- On veut créer un lien logique et physique entre des variables
- On veut désigner sous un seul nom un ensemble de valeurs/variables/champs pouvant être de types différents
- On veut faciliter le passage d'arguments dans les sous-programmes
- On veut faciliter l'utilisation de l'allocation dynamique

Exemple :

- Nom, adresse, solde d'un **client** d'une banque.
- Clients d'une **banque**.
- Partie réelle et imaginaire d'un **nombre complexe**.

Structurer l'environnement = Structurer l'accès à l'information

Algorithmique

Création d'une structure

client

nom
prenom
valeur

Une structure est comme un « panier » contenant différents objets (ou champs) ayant un lien logique entre eux

FORTRAN

! définition

```
TYPE client
  character (len=20) :: nom
  character (len=10) :: prenom
  real :: valeur
END TYPE client
```

Algorithmique

Déclaration d'une structure

On « fait appel » à un champ de la structure en utilisant le %

Ex.: on affecte à l'élément « nom » de la structure « cl1 » la valeur « Bonometti »

FORTRAN

! déclaration de variables

```
TYPE (client) :: cl1
```

! accès à un champ et affectation

```
cl1%nom = "Bonometti"
```

```
cl1%prenom = "Thomas"
```

```
cl1%valeur = 142857.0
```

Exemple de programme simple avec des structures

```
PROGRAM FACTURE
  IMPLICIT NONE

  type article
    INTEGER :: n
    REAL    :: pht
  end type article

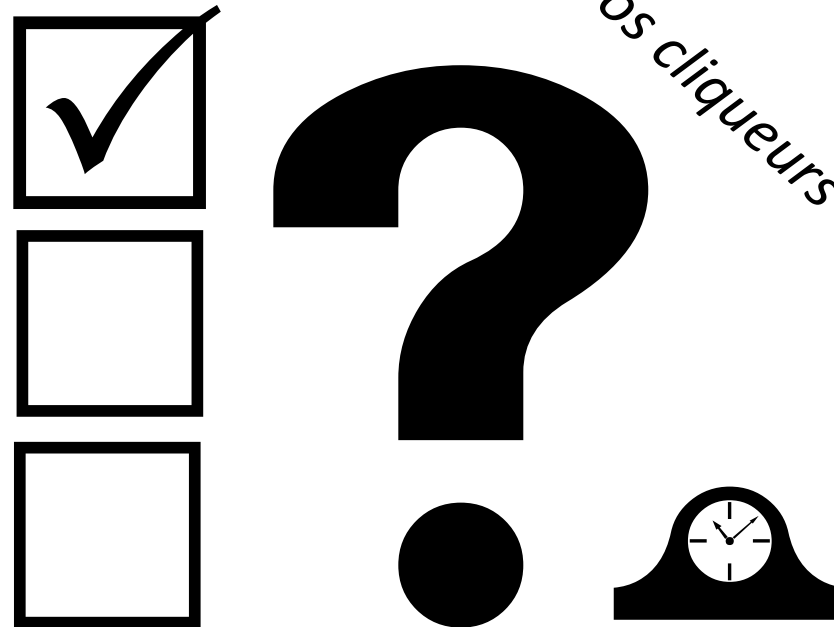
  type(article) :: a1, a2
  REAL          :: p1, p2, pttc

  WRITE(*,*) "Prix HT unitaire du 1er type d'articles ?"
  READ(*,*) a1%pht
  WRITE(*,*) "Nombre d'articles du 1er type ?"
  READ(*,*) a1%n
  p1 = real(a1%n) * a1%pht

  WRITE(*,*) "Prix HT unitaire du 2eme type d'articles ?"
  READ(*,*) a2%pht
  WRITE(*,*) "Nombre d'articles du 2eme type ?"
  READ(*,*) a2%n
  p2 = real(a2%n) * a2%pht

  pttc = (p1+p2) * 1.186
  WRITE(*,*) "La facture est de: ", pttc, " euros"

END PROGRAM FACTURE
```



Quelle version est correcte ?

Quiz

A

```
program toto
  implicit none

  type personne
    real :: taille, poids
  end type personne

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, taille%p1, poids%p1
  imc = poids%p1 / taille%p1**2
  print*,"Votre IMC vaut:",imc

end program toto
```

B

```
program toto
  implicit none

  type personne
    real :: taille, poids
  end type personne

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, taille, poids
  imc = poids / taille**2
  print*,"Votre IMC vaut:",imc

end program toto
```

C

```
program toto
  implicit none

  type personne
    real :: taille, poids
  end type personne

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, p1%taille, p1%poids
  imc = p1%poids / p1%taille**2
  print*,"Votre IMC vaut:",imc

end program toto
```

D

```
program toto
  implicit none

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, p1%taille, p1%poids
  imc = p1%poids / p1%taille**2
  print*,"Votre IMC vaut:",imc

end program toto
```

Quelle version est correcte ?

Quiz

A

```
program toto
  implicit none

  type personne
    real :: taille, poids
  end type personne

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, taille%p1, poids%p1
  imc = poids%p1 / taille%p1**2
  print*,"Votre IMC vaut:",imc

end program toto
```

B

```
program toto
  implicit none

  type personne
    real :: taille, poids
  end type personne

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, taille, poids
  imc = poids / taille**2
  print*,"Votre IMC vaut:",imc

end program toto
```

#QDLE#Q#ABC*D#30#

C

```
program toto
  implicit none

  type personne
    real :: taille, poids
  end type personne

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, p1%taille, p1%poids
  imc = p1%poids / p1%taille**2
  print*,"Votre IMC vaut:",imc

end program toto
```

D

```
program toto
  implicit none

  type(personne) :: p1
  real :: imc

  print*,"taille (m) et poids (kg) ?"
  read*, p1%taille, p1%poids
  imc = p1%poids / p1%taille**2
  print*,"Votre IMC vaut:",imc

end program toto
```


Algorithmique

Soit la structure « vect » contenant 3 entiers (n, m, l), 3 réels (t, val1, val2) et un tableau de rang 1 et d'étendue 100 (u)

Le champ n de la structure v1 reçoit 5

Le vecteur u de v1 est nul

u de la structure v1 <- u de la structure v2

v2 reçoit toutes les valeurs des champs de v1

FORTRAN

définition

```
TYPE vect
  INTEGER :: n, m, l
  REAL :: t, val1, val2
  REAL, DIMENSION(100) :: u
END TYPE vect
```

déclaration

```
TYPE (vect) :: v1, v2
```

affectation

```
v1%n = 5
v1%u(:) = 0.
v1%u(:) = v2%u(:)
v2 = v1
```

Algorithmique

Soit la structure « matrice_carre »
contenant un tableau de rang 1 (u) et un
tableau de rang 2 (m)

Remarque: éviter de donner le même nom à
une structure et le champ de la structure
(même si cela est possible...)

Exemple de manipulation
des données de la structure

FORTRAN

définition

```
TYPE matrice_carre
  REAL, DIMENSION(100) :: u
  REAL, DIMENSION(100,100) :: m
END TYPE matrice_carre
```

déclaration et manipulations

```
TYPE (matrice_carre) :: m1
INTEGER :: i,j

DO i = 1,100
  DO j = 1,100
    m1%m(i,j) = real(j) * m1%u(i)
  END DO
END DO
```

Algorithmique

date

| jour
| mois
| an

client

| nom
| prenom
| date_de_naissance
| compte

FORTRAN

Exercice:

définir les différentes structures
(avec plusieurs variantes...)

Exercice

- Ecrire le programme demandant à l'utilisateur sa date de naissance et affectant le résultat dans le champ correspondant de la structure « client » définie précédemment (avec plusieurs variantes...)

Quand utiliser une structure?

```
PROGRAM TOTO_1
```

```
  IMPLICIT NONE
  ! Declarations
  INTEGER :: i,j,k,l,m,n
  REAL :: x,y,z
  REAL, DIMENSION (100) :: u,v,w

  ! Instructions
  CALL lecture(m,u)
  CALL initialisation(m,v)
  CALL calcul (u,v,w)
```

```
END PROGRAM TOTO_1
```

```
PROGRAM TOTO_2
```

```
  IMPLICIT NONE
  ! Declarations
  INTEGER :: i,j,k,l,m,n
  REAL :: x,y,z
  REAL, DIMENSION (100) :: u,v,w

  ! Instructions
  CALL lecture(i,j,k,l,m,n,x,y,z,u,v)
  CALL initialisation(i,j,k,l,m,n,x,y,z,u,v,w)
  CALL calcul (l,m,n,x,y,z,u,v,w)
```

```
END PROGRAM TOTO_2
```

Utiliser une structure doit simplifier la programmation, et non l'inverse !

Blanc bonnet \neq bonnet blanc ...

Structure contenant un tableau

définition

```
TYPE vect
  INTEGER :: n
  REAL, DIMENSION(100) :: u
END TYPE vect
```

Déclaration et manipulation

```
TYPE (vect) :: v1
INTEGER :: i
```

```
DO i = 1,100
  v1%u(i) = 0.
END DO
```

s'applique au champ
de la structure

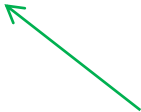


Tableau de structure

définition

```
TYPE client
  CHARACTER (len=20) :: nom
  REAL :: valeur
END TYPE client
```

Déclaration et manipulation

```
TYPE (client), DIMENSION(50) :: b1
INTEGER :: i
```

```
DO i = 1,50
  b1(i)%valeur = 0.
END DO
```

s'applique à
la structure



Tri

Fonctionnalité : On veut trier par ordre croissant les éléments d'un vecteur contenant n éléments ($n \leq 100$), stockés dans un fichier texte nommé `don.dat` et écrire le résultats dans un fichier texte appelé `res.dat`.

On utilise pour cela:

- Une structure qui a pour champs le nombre d'éléments du vecteur et le vecteur lui-même
- Des tableaux dynamiques

Exemple du tri (6)

```
PROGRAM TRI
  IMPLICIT NONE

  ! DEFINITION DES STRUCTURES
  TYPE struct
    INTEGER :: n
    REAL, DIMENSION(:), ALLOCATABLE :: t
  END TYPE struct

  ! DECLARATIONS
  TYPE (struct) :: u, v

  ! INSTRUCTIONS
  CALL lecture_valeur('don.dat',u%n)
  v%n = u%n
  CALL lecture_vecteur('don.dat',u)
  CALL tri_vecteur(u,v)
  DEALLOCATE(u%t)
  CALL ecriture('res.dat',v)
  DEALLOCATE(v%t)
END PROGRAM TRI
```

```
!-----
SUBROUTINE lecture_valeur(nomfich,n)

  IMPLICIT NONE
  ! DECLARATIONS
  ! variables en arguments
  CHARACTER(len=7), INTENT(IN) :: nomfich
  INTEGER, INTENT(OUT) :: n
  ! variable locale
  INTEGER :: i

  ! INSTRUCTIONS
  OPEN(10, file=nomfich)
  READ(10,*) n
  CLOSE(10)

END SUBROUTINE lecture_valeur
```

champ mis en argument

désallocation

structure mis en argument

Exemple du tri (6)

```
!-----  
SUBROUTINE lecture_vecteur(nomfich,u)  
  
  IMPLICIT NONE  
  ! DEFINITION DES STRUCTURES  
  TYPE struct  
    INTEGER :: n  
    REAL, DIMENSION(:), ALLOCATABLE :: t  
  END TYPE struct  
  ! DECLARATIONS  
  ! variables en arguments  
  CHARACTER(len=7), INTENT(IN) :: nomfich  
  TYPE (struct), intent(INOUT) :: u  
  ! variable locale  
  INTEGER :: i  
  
  ! INSTRUCTIONS  
  ALLOCATE(u%t(u%n))  
  OPEN(10, file=nomfich)  
  READ(10,*) i  
  DO i = 1, u%n  
    READ(10,*) u%t(i)  
  END DO  
  CLOSE(10)  
  
END SUBROUTINE lecture_vecteur
```

↑
IN pour garder la
valeur de u%n

← allocation →

```
!-----  
SUBROUTINE tri_vecteur(u,v)  
  
  IMPLICIT NONE  
  ! DEFINITION DES STRUCTURES  
  TYPE struct  
    INTEGER :: n  
    REAL, DIMENSION(:), ALLOCATABLE :: t  
  END TYPE struct  
  ! DECLARATIONS  
  ! variables en arguments  
  TYPE (struct), intent(INOUT) :: u  
  TYPE (struct), intent(INOUT) :: v  
  ! fonctions  
  REAL :: val_max  
  ! variable locale  
  REAL :: umax, umin  
  INTEGER :: i, jmin  
  
  ! INSTRUCTIONS  
  ALLOCATE(v%t(v%n))  
  umax = val_max(u%n,u%t)  
  DO i = 1, u%n  
    CALL recherche_min(u%n,u%t,umax,umin,jmin)  
    v%t(i) = umin  
    u%t(jmin) = umax  
  END DO  
  
END SUBROUTINE tri_vecteur
```

Exemple du tri (6)

```
!-----  
FUNCTION val_max(k,p)  
  
  IMPLICIT NONE  
  ! DECLARATIONS  
  ! variables en argument  
  INTEGER, INTENT(IN) :: k  
  REAL, DIMENSION(k), INTENT(IN) :: p  
  ! fonction  
  REAL :: val_max  
  ! variables locales  
  INTEGER :: i  
  
  ! INSTRUCTIONS  
  val_max = p(1)  
  DO i = 2, k  
    IF( p(i) >= val_max ) THEN  
      val_max = p(i)  
    END IF  
  END DO  
  
END FUNCTION val_max
```

```
!-----  
SUBROUTINE recherche_min(n,u,umax,umin,jmin)  
  
  IMPLICIT NONE  
  ! DECLARATIONS  
  ! variables en arguments  
  INTEGER, INTENT(IN) :: n  
  REAL, DIMENSION(n), INTENT(IN) :: u  
  REAL, INTENT(IN) :: umax  
  REAL, INTENT(OUT) :: umin  
  INTEGER, INTENT(OUT) :: jmin  
  ! variables locales  
  INTEGER :: i  
  
  ! INSTRUCTIONS  
  jmin = 1  
  umin = umax  
  DO i = 1, n  
    IF ( u(i) <= umin ) THEN  
      umin = u(i)  
      jmin = i  
    END IF  
  END DO  
  
END SUBROUTINE recherche_min
```

Exemple du tri (6)

```
!-----  
SUBROUTINE ecriture(nomfich,v)  
  
    IMPLICIT NONE  
    ! DEFINITION DES STRUCTURES  
    TYPE struct  
        INTEGER :: n  
        REAL, DIMENSION(:), ALLOCATABLE :: t  
    END TYPE struct  
    ! DECLARATIONS  
    ! variables en arguments  
    CHARACTER(len=7), INTENT(IN) :: nomfich  
    TYPE (struct), intent(IN) :: v  
    ! variable locale  
    INTEGER :: i  
  
    ! INSTRUCTIONS  
    OPEN(10, file=nomfich)  
    WRITE(10,*) v%n  
    DO i = 1, v%n  
        WRITE(10,*) v%t(i)  
    END DO  
    CLOSE(10)  
  
END SUBROUTINE ecriture
```

Suite de Fibonacci

- Ecrire une structure « toto » contenant un entier et un tableau de rang 1 et d'étendue 100 fixée

- En utilisant la structure toto, écrire un programme **sans procédures** qui détermine l'ensemble des valeurs u_1, u_2, \dots, u_n ($n \leq 100$ étant fourni en donnée) de la suite définie comme suit: $u_1 = 1, u_2 = 1, u_n = u_{n-1} + u_{n-2}$ (pour $n > 2$). Les valeurs n, u_1 et u_2 sont stockées dans le fichier formaté don_entree.dat sous la forme: « n u₁ u₂ » sur une ligne. Les résultats seront écrits dans le fichier texte don_sortie.dat sous la forme:

« Les 'n' premiers termes de la suite de Fibonacci sont:

u(1)

...

u(n) »

