INP TOULOUSE ENSEEIHT

Programmation impérative: Méthode de Programmation

6. Modules + Formats

Table des matières

INP TOULOUSE ENSEEIHT

Plan du cours

Modules

Modules

Unité autonome pouvant être utilisée au sein de n'importe quelle autre, comme si l'on avait explicité son contenu

Intérêts:

- éviter la duplication de déclarations identiques
- partager des données
- optimiser la mémoire (un module pour chaque 'option')
- faciliter l'utilisation de l'allocation dynamique

Création d'un module

Algorithmique

Création d'un module appelé « m_type » contenant deux structures:

- « client »
- « point »

FORTRAN

```
! définition
MODULE m_type
IMPLICIT NONE

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

TYPE point
  integer :: num
  real :: x, y
END TYPE point

END MODULE m_type
```

Utilisation des variables d'un module

Algorithmique

Exemple d'un programme, dont on veut pouvoir utiliser les variables de type « client » et « point »

Avec un module, plus besoin de définir chaque structure dans chaque programme /procédure

→ gain de temps + moins d'erreurs

FORTRAN

```
PROGRAM toto
USE m_type
IMPLICIT NONE

! Declarations
TYPE (client) :: cl1, cl2
TYPE (point) :: p1, p2

! Instructions
(…)

END PROGRAM toto
```

Avant le IMPLICIT NONE

Exemple de programme simple avec un module

```
MODULE mon_module

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

END MODULE mon_module

PROGRAM FACTURE
   use mon_module
   IMPLICIT NONE

   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
```

Exemple du tri (7)

Tri

Fonctionnalité : On veut trier par ordre croissant les éléments d'un vecteur contenant n éléments (n<=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
- La structure est mise dans un module appelé « m_type »
- Des tableaux dynamiques

Exemple du tri (7)

```
MODULE m_type
   IMPLICIT NONE

   TYPE struct
       INTEGER :: n
       REAL, DIMENSION(:), ALLOCATABLE :: t
   END TYPE struct

END MODULE m_type

!----------
PROGRAM TRI

   ! APPEL DU MODULE
   USE m_type

   IMPLICIT NONE

   ! 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
```

Exemple du tri (7)

```
!---------------------------------------
SUBROUTINE lecture_vecteur(nomfich,u)

   ! APPEL DU MODULE
   USE m_type

   IMPLICIT NONE

   ! 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
```

```
!----------------------------
SUBROUTINE tri_vecteur(u,v)

   ! APPEL DU MODULE
   USE m_type

   IMPLICIT NONE

   ! 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 (7)

```
!--------------------
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 (7)

```
!-------------------------------
SUBROUTINE ecriture(nomfich,v)

   ! APPEL DU MODULE
   USE m_type

   IMPLICIT NONE

   ! 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
```

Formats

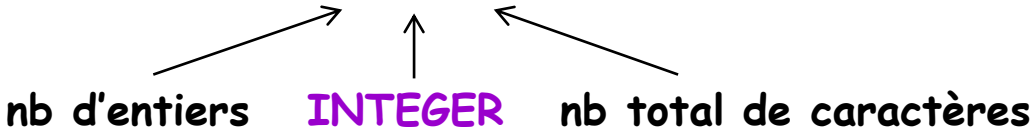
Intérêts: - contrôle des données écrites/lues
- lisibilité des données affichées
- lecture de plusieurs chaines de caractères

Exemple sans format: WRITE(*,*)
Exemple avec format: WRITE(*,'(i5)')

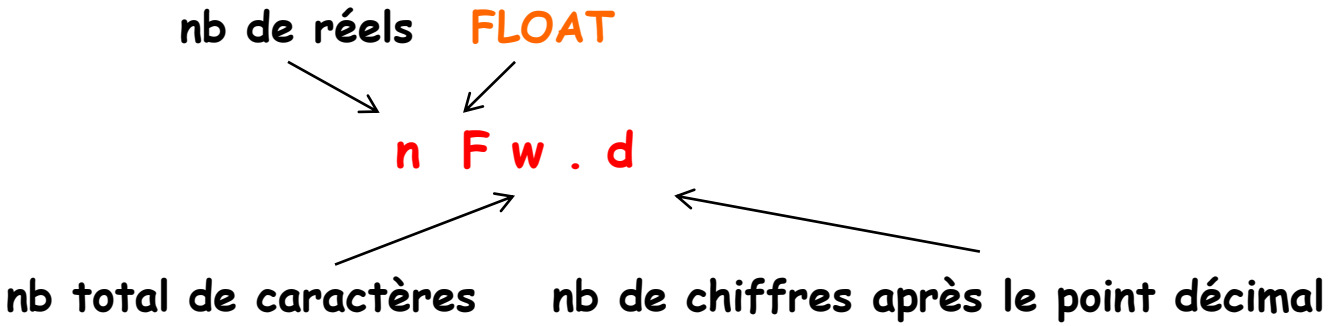
Descripteurs de formats

Entiers: **n I w**

Espace: **X**



Réels: **n F w . d**



Réels (suite): **n E w . d**

EXPONENTIAL

Chaîne de caractères: **A w**

Exemple de programme simple avec un format

```
MODULE mon_module

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

END MODULE mon_module

PROGRAM FACTURE
   use mon_module
   IMPLICIT NONE

   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(*,'(f7.1)') "La facture est de: ", pttc, " euros"

END PROGRAM FACTURE
```

Exemples

```
PROGRAM formats_01

   IMPLICIT NONE
   INTEGER :: i,j

   i = 2013
   j = 123456

   WRITE(*,*) i
   WRITE(*,'(i6)') i
   WRITE(*,'(i5)') i
   WRITE(*,'(i4)') i
   WRITE(*,'(i3)') i

   WRITE(*,*) i, j
   WRITE(*,'(2i8)') i, j
   WRITE(*,'(2i7)') i, j
   WRITE(*,'(2i6)') i, j
   WRITE(*,'(2i5)') i, j

   WRITE(*,'(i6,1x,i6)') i, j
   WRITE(*,'(2(i6,1x))') i, j
   WRITE(*,100) i, j
100 FORMAT(2(i6,1x))

END PROGRAM formats_01
```

AFFICHAGE

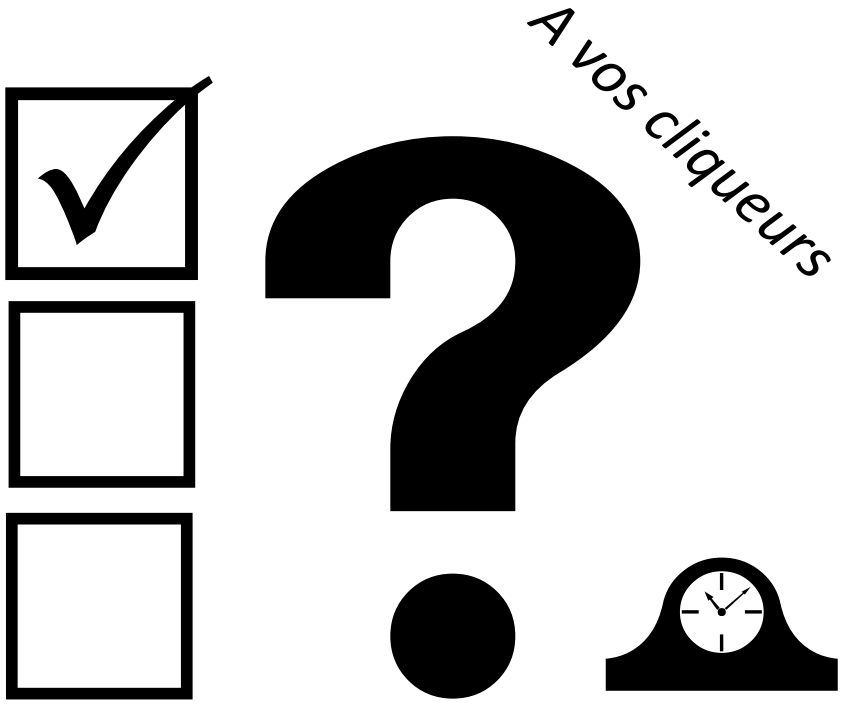
```
tbonomet@perche:~/FORTRAN90$ ./prog.exe
        2013
  2013
 2013
2013
***
        2013      123456
    2013  123456
   2013 123456
  2013123456
 2013*****
  2013 123456
  2013 123456
  2013 123456
tbonomet@perche:~/FORTRAN90$
```

Exemples

```
PROGRAM format_02

    IMPLICIT NONE
    INTEGER :: i,j
    REAL(KIND=8) :: x

    x = 3.1415927

    WRITE(*,*) x
    WRITE(*,'(f10.6)') x
    WRITE(*,'(f10.7)') x
    WRITE(*,'(f10.8)') x
    WRITE(*,'(f10.9)') x

    WRITE(*,'(e15.8)') x
    WRITE(*,'(e15.4)') x
    WRITE(*,'(e12.8)') x

    i = 2013
    j = 123456
    WRITE(*,*) i, j, x
    WRITE(*,200) i, j, x
200 FORMAT("i et j valent respectivement ",i4,1x,i6," et x =",f10.7)

END PROGRAM format_02
```

AFFICHAGE

```
tbonomet@perche:~/FORTRAN90$ ./prog.exe
   3.1415927410125732
  3.141593
 3.1415927
3.14159274
**********
 0.31415927E+01
     0.3142E+01
************
        2013      123456   3.1415927410125732
i et j valent respectivement 2013 123456 et x = 3.1415927
tbonomet@perche:~/FORTRAN90$
```


A vos cliqueurs

Quiz n°1

Quoi mettre à la place du ? pour pouvoir afficher le nombre avec cinq chiffres après la virgule en format « float »

```
PROGRAM  nb_pi

   IMPLICIT NONE
   REAL :: x

   x = -1. * acos(-1.)
   WRITE(*,?) x

END PROGRAM  nb_pi
```

A. '(f6.5)'

B. 'f7.5'

C. '(f7.5)'

D. 'f7.5'

E. '(f8.5)'

F. ('f9.5')

#QDLE#Q#ABCDE*F#60#

Quiz n°2

Quoi mettre à la place des ? pour afficher sur une même ligne les 30 éléments du vecteur avec 5 chiffres après la virgule en format « exponential »

```
PROGRAM  i2

   IMPLICIT NONE

   REAL, DIMENSION(30) :: u
   INTEGER :: i

   DO i=1,30
      u(i) = real(i)**2
   END DO

   WRITE(*, ? ) ?

END PROGRAM  i2
```

A. `WRITE(*,'(30E10.5)') (u(i),i=1,30)`

B. `WRITE(*,'(30(E10.5,1X))') (u(i),i=1,30)`

C. `WRITE(*,'(30E11.5)') (u(i),i=1,30)`

D. `WRITE(*,'(30(E11.5,1X))') (u(i),i=1,30)`

E. Rien de tout cela

#QDLE#Q#ABCD*E#60#

Chercher les 10 erreurs

Quiz n°3

```
module mod_tb

  type personne
    character (len=15) :: nom
    real :: taille,poids,imc
  end type personne

end module mod_tb

program main_imc

   use mod_tb
   implicite none
   type(personne), dimension(:), allocatable :: p
   integer :: np

   print*,"Entrer le nombre de personnes"
   read*,np
   allocate(p(np))
   do i=0,np
     call calcul_imc(p(i))
     write(10,*) p(i)%nom,p%imc(i)
   end do

10 format ("L'IMC de ",a15,1x," vaut: ",f4.1)

end program main_imc
```

```
subroutine calcul_imc(np,q)


  implicit none
  type(personne), intent(in) :: q

  print*,"nom, taille(m), poids(kg) ?"
  read*,q%nom,q%taille,q%poids
  imc%q = q%poids / q%taille**2

end subroutine calcul_imc
```

Chercher les 10 erreurs

Quiz n°3

Exemple de programme utilisant allocation dynamique, subroutine, module, structure, formats

```
module mod_tb

  type personne
    character (len=15) :: nom
    real :: taille,poids,imc
  end type personne

end module mod_tb

program main_imc

   use mod_tb
   implicit none
   type(personne), dimension(:), allocatable :: p
   integer :: i,np

   print*,"Entrer le nombre de personnes"
   read*,np
   allocate(p(np))
   do i=1,np
     call calcul_imc(p(i))
     write(*,10) p(i)%nom,p(i)%imc
   end do
   deallocate(p)
10 format ("L'IMC de ",a15,1x," vaut: ",f4.1)

end program main_imc
```

```
subroutine calcul_imc(q)

  use mod_tb
  implicit none
  type(personne), intent(out) :: q

  print*,"nom, taille(m), poids(kg) ?"
  read*,q%nom,q%taille,q%poids
  q%imc = q%poids / q%taille**2

end subroutine calcul_imc
```

Création de fichiers de nom variable : un exemple

FORTRAN

```
PROGRAM fichiers_nom_var
   IMPLICIT NONE

   INTEGER :: i
   CHARACTER(LEN=3) :: num
   REAL (KIND=8) :: x

    x = acos(-1.)

    DO i=1,10
        WRITE(num,"(i3.3)") i
        OPEN(10,file = "res_"//num//".dat")
        WRITE(10,*) x
        CLOSE(10)
    END DO

END PROGRAM fichiers_nom_var
```

AFFICHAGE

```
>
> ls
> fichiers_nom_var.f90
> gfortran fichiers_nom_var.f90 –o prog.exe
> ls
> fichiers_nom_var.f90    prog.exe
> ./prog.exe
> ls
> fichiers_nom_var.f90    prog.exe res_001.dat
res_002.dat res_003.dat res_004.dat
res_005.dat res_006.dat res_007.dat
res_008.dat res_009.dat  res_010.dat
>
```

Fichiers non-formatés

Quel types de fichiers pour quels type d'accès?

Entrée - Sortie	Formatée	Non Formatée
Séquentiel	X On sait comment on a écrit dans le fichier	
Direct		X On sait où on a écrit dans le fichier

Remarque:

- Lire/écrire en direct dans des fichiers formatés est possible (mais moins utilisé)
- Lire/écrire en séquentiel dans des fichiers non-formatés est possible (mais moins utilisé)

Ecriture dans un fichier (direct / non-formaté)

- Pour l'accès direct, tous les « enregistrements » doivent avoir la même taille
- Il faut déterminer la « taille » des enregistrements

```
program ecriture_non_formate

 implicit none
 REAL, DIMENSION(100) ::  v
 INTEGER :: i, long

  do i=1,100
    v(i)=real(i)
  end do

  INQUIRE(IOLENGTH=long) v
  OPEN (10, file = "toto.dat", form = "unformatted", access = "direct", recl=long)
  DO i = 1, 100
     WRITE(10,rec=i) v(i)
  END DO
  CLOSE (10)

end program ecriture_non_formate
```

Ici, on récupère la taille de la donnée

Ici, on renseigne la taille de la donnée

Ici, on positionne la donnée dans le fichier

Lecture dans un fichier (direct / non-formaté)

```
program lecture_non_formate

 implicit none
 REAL, DIMENSION(100) ::  v
 INTEGER :: i, long

  INQUIRE(IOLENGTH=long) v
  OPEN (10, file = "toto.dat", form = "unformatted", access = "direct", recl=long)
  DO i = 1, 100
     READ(10,rec=i) v(i)
  END DO
  CLOSE (10)

  open(20,file="toto_relu_formate.dat")
  do i = 1,100
     write(20,*) i, v(i)
  end do
  close(20)

end program lecture_non_formate
```

Pour conclure, quelques conseils de programmation

- Ecrire un code qui soit clair pour le lecteur et le compilateur
- Commenter les sources
- Donner des noms aux procédures qui ont un sens
- Ranger
- Dupliquer les sources quand une version fonctionne.

Ce qui n'est pas fait

- Notion d'interfaces
- Tableaux automatiques
- Les notions de listes
- Les pointeurs