INP TOULOUSE ENSEEIHT

Programmation impérative:
Méthode de Programmation

4. Compilation avec Makefile + Allocation dynamique

Table des matières

INP TOULOUSE ENSEEIHT

Plan du cours

Compilation via un fichier Makefile

Mise en œuvre d'un algorithme

OPTION #1

tout dans 1 fichier (OK si programme court)

OPTION #2

3 fichiers

1°) le programme principal,

2°) le prototypage des structures

3°) l'implémentation des différents sous-programmes.

OPTION #3

N fichiers (prog + 1/ss-prog + 1/structure)

OPTION #1 (rappel)

Programmation

```
! Code source dans le fichier « prog.f90 »
PROGRAM COUCOU
  IMPLICIT NONE
  PRINT*, 'Coucou'
END PROGRAM COUCOU
```

Compilation (lignes de commandes)

```
! Compilation
> ls
> prog.f90
> gfortran prog.f90 –o coucou.exe
> ls
> prog.f90   coucou.exe
```

Exécution

```
! Exécution
> ./coucou.exe
> Coucou
```

OPTION #2 et #3 → Makefile

Programmation

```
! Ex.:
! Programme principal dans le fichier « prog.f90 »
! Sous-programmes dans « sousprog.f90 »
! Structures dans « m_type.f90 »
```

Compilation (Makefile)

```
! Compilation
> ls
> prog.f90  sousprog.f90   m_type.f90   Makefile
> make
(…)
> ls
> prog.f90  sousprog.f90   m_type.f90  prog.o
sousprog.o   m_type.o   Makefile   coucou.exe
```

Exécution

```
! Exécution
> ./coucou.exe
> Coucou
```

Exemple de fichier Makefile

Makefile

```
FC = gfortran
OPT = -g –O0 –fbounds-check

OBJ = m_type.o prog.o sousprog.o
EXE = coucou.exe

coucou:    $(OBJ)
     $(FC) $(OPT) $(OBJ) -o $(EXE)

m_type.o:   m_type.f90
     $(FC) $(OPT) m_type.f90 -c

prog.o :    prog.f90
     $(FC) $(OPT) prog.f90 -c

sousprog.o :          sousprog.f90
     $(FC) $(OPT) sousprog.f90 -c
```

Ordre à respecter

```
OBJ = 1 module.o
      2 programme_principal.o
      3 subroutines.o
```

```
1     module.o: …
```

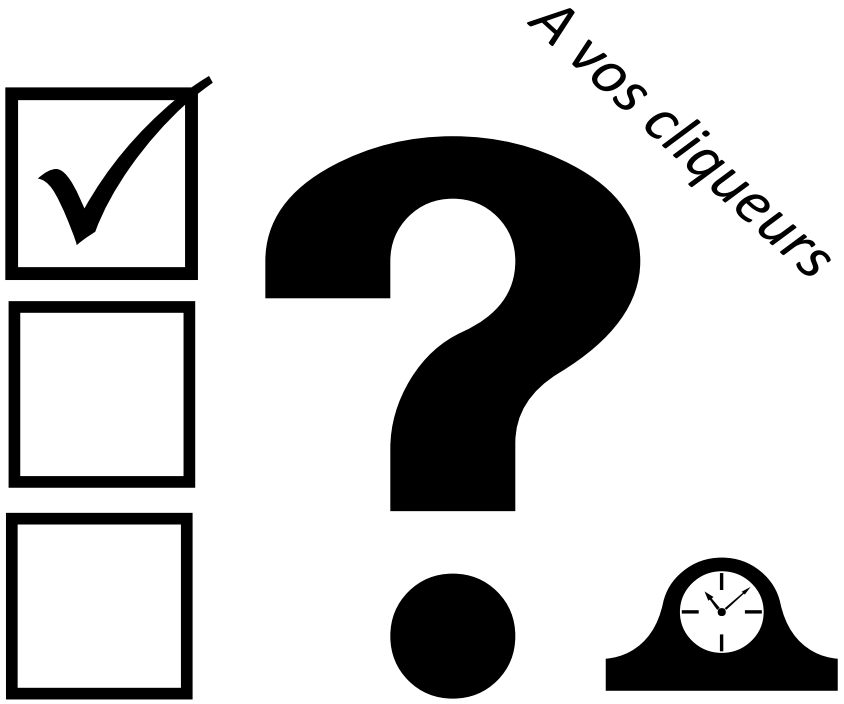
```
2     programme_principal.o: …
```

```
3     subroutines.o: …
```

INP TOULOUSE ENSEEIHT

Plan du cours

Allocation dynamique


A vos cliqueurs

Quiz n°1

Que fait ce programme pour nl=nc=2 ?

```
PROGRAM matrice_aij

IMPLICIT NONE
REAL, DIMENSION(100,100) :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO

END PROGRAM matrice_aij
```

A. Affiche: 2 4 3 6

B. Affiche: 2
4
3
6

C. Rien…

D. Affiche: 2 4
3 6

E. Affiche: 2 3
4 6

Quiz n°1

Que fait ce programme pour nl=nc=2 ?

```
PROGRAM matrice_aij

IMPLICIT NONE
REAL, DIMENSION(100,100) :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO

END PROGRAM matrice_aij
```

A. Affiche: 2 4 3 6

B. Affiche:
2
4
3
6

C. Rien…

D. Affiche:
2 4
3 6

E. Affiche:
2 3
4 6

Quiz n°2

Pour nl = nc = 100, il y a :

```
PROGRAM matrice_aij

IMPLICIT NONE
REAL, DIMENSION(100,100) :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO

END PROGRAM matrice_aij
```

A. Une erreur à la compilation

B. Une erreur à l'exécution

C. Une exécution OK

D. Une exécution OK
mais ce n'est pas optimal…

#QDLE#Q#ABC*D#45#

Quiz n°3

Pour nl = 90, nc = 110, il y a :

```
PROGRAM matrice_aij

IMPLICIT NONE
REAL, DIMENSION(100,100) :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO

END PROGRAM matrice_aij
```

A. Une erreur à la compilation

B. Une erreur à l'exécution

C. Une exécution OK

D. Une exécution OK
mais ce n'est pas optimal…

#QDLE#Q#AB*CD#30#

Quiz n° 4

Pour nl = nc = 10, il y a :

```
PROGRAM matrice_aij

IMPLICIT NONE
REAL, DIMENSION(100,100) :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO

END PROGRAM matrice_aij
```

A. Une erreur à la compilation

B. Une erreur à l'exécution

C. Une exécution OK

D. Une exécution OK
mais ce n'est pas optimal…

#QDLE#Q#ABCD*#30#

Allocation dynamique, pour quoi faire ?

POUR OPTIMISER LES RESSOURCES DE MÉMOIRE

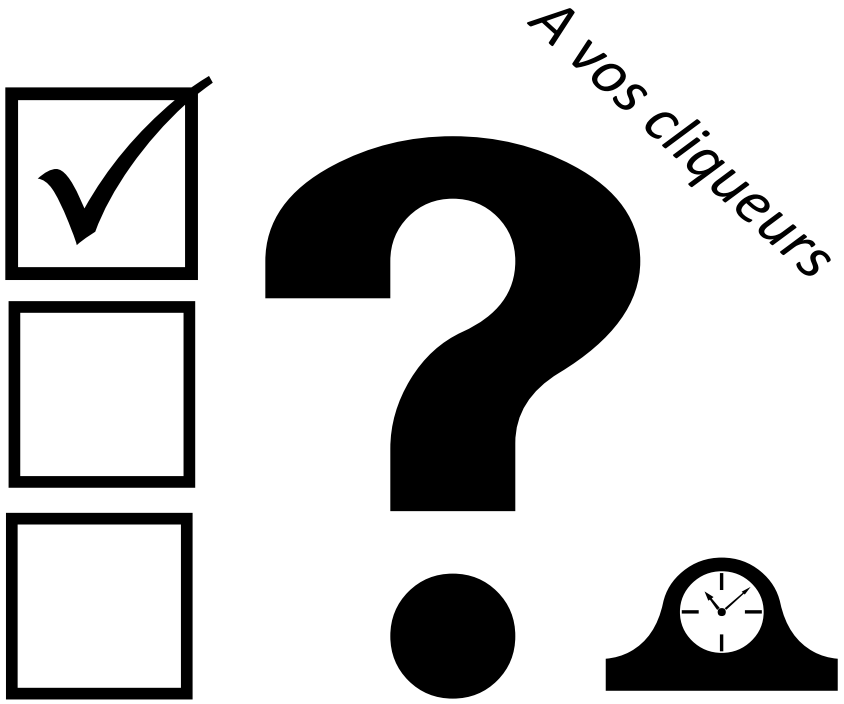
- en n'allouant que l'espace nécessaire

- en n'allouant que lorsque c'est nécessaire

- en libérant la mémoire utilisée pour une variable dès que celle-ci devient inutile

Allocation dynamique

Algorithmique	FORTRAN
Création d'un tableau de rang 1, dont le nombre d'éléments est inconnu au moment de la création	REAL, DIMENSION(**:**), **ALLOCATABLE** :: v
Allocation de l'espace mémoire optimal quand le nombre d'éléments est connu	**ALLOCATE**(v(**n**))
Libération de l'espace mémoire après utilisation	**DEALLOCATE**(v)


A vos cliqueurs

Pour nl = 900, nc = 10, il y a :

```
PROGRAM matrice_aij

IMPLICIT NONE
REAL, DIMENSION(:,:), ALLOCATABLE :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nombre de lignes et de colonnes'
READ*, nl, nc
ALLOCATE (A(nl,nc))

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO
DEALLOCATE (A)

END PROGRAM matrice_aij
```

A. Une erreur à la compilation

B. Une erreur à l'exécution

C. Une exécution OK et c'est optimal

D. Une exécution OK mais ce n'est pas optimal…

#QDLE#Q#ABC*D#30#

Chercher l'erreur… 1/2

```
PROGRAM matrice_aij_01

IMPLICIT NONE
REAL, DIMENSION(:,:), ALLOCATABLE :: A
INTEGER :: i, j, nl, nc

ALLOCATE (A(nl,nc))
PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO
DEALLOCATE (A)

END PROGRAM matrice_aij_01
```

```
PROGRAM matrice_aij_02

IMPLICIT NONE
REAL, DIMENSION(:,:), ALLOCATABLE :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc
ALLOCATE (A(nl,nc))
DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DEALLOCATE (A)
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO

END PROGRAM matrice_aij_02
```

Chercher l'erreur… 2/2

```
PROGRAM matrice_aij_03

IMPLICIT NONE
REAL, DIMENSION(:,:), ALLOCATABLE :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc

DO i = 1,nl
   DO j = 1,nc
      ALLOCATE (A(nl,nc))
      A(i,j) = (i+1)*j
      DEALLOCATE (A)
   END DO
END DO
DO i = 1,nl
  ALLOCATE (A(nl,nc))
  PRINT*, (A(i,j), j=1,nc)
  DEALLOCATE (A)
END DO

END PROGRAM matrice_aij_03
```

```
PROGRAM matrice_aij_04

IMPLICIT NONE
REAL, DIMENSION(:,:), ALLOCATABLE :: A
INTEGER :: i, j, nl, nc

PRINT*, 'nb de lignes et de colonnes'
READ*, nl, nc
ALLOCATE (A(nc,nl))
DO i = 1,nl
   DO j = 1,nc
      A(i,j) = (i+1)*j
   END DO
END DO
DO i = 1,nl
  PRINT*, (A(i,j), j=1,nc)
END DO
DEALLOCATE (A)

END PROGRAM matrice_aij_04
```

FORTRAN

Les tableaux dynamiques, qui ne sont ni dans une structure ni dans un module, doivent être alloués et libérés dans la même unité de programmation

Exemple du tri (5)

Tri

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

- Pour cela, on utilise, pour optimiser les ressources en mémoire, l'allocation dynamique

Exemple du tri (5)

```
PROGRAM TRI
   IMPLICIT NONE
   ! DECLARATIONS
   INTEGER :: n
   REAL, DIMENSION(:), ALLOCATABLE :: u, v

   ! INSTRUCTIONS
   CALL lecture_valeur('don.dat',n)
   ALLOCATE(u(n),v(n))
   CALL lecture_vecteur('don.dat',n,u)
   CALL tri_vecteur(n,u,v)
   DEALLOCATE(u)
   CALL ecriture('res.dat',n,v)
   DEALLOCATE(v)

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
```

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

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

   ! INSTRUCTIONS
   OPEN(10, file=nomfich)
   READ(10,*) i
   DO i = 1, n
      READ(10,*) u(i)
   END DO
   CLOSE(10)

END SUBROUTINE lecture_vecteur
```

Exemple du tri (5)

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

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(n), INTENT(INOUT)  :: u
   REAL, DIMENSION(n), INTENT(OUT) :: v
   ! fonctions
   REAL :: val_max
   ! variable locale
   REAL :: umax, umin
   INTEGER :: i, jmin

   ! INSTRUCTIONS
   umax = val_max(n,u)
   DO i = 1, n
        CALL recherche_min(n,u,umax,umin,jmin)
        v(i) = umin
        u(jmin) = umax
   END DO

END SUBROUTINE tri_vecteur
```

```
!--------------------
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
```

Exemple du tri (5)

```
!-----------------------------------------
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
```

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

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

   ! INSTRUCTIONS
   OPEN(10, file=nomfich)
   WRITE(10,*) n
   DO i = 1, n
      WRITE(10,*) v(i)
   END DO
   CLOSE(10)

END SUBROUTINE ecriture
```

Exercices

Ecrire un programme qui calcule un vecteur u de dimension n (n>=1 étant fourni en donnée à l'écran) tel que $u(i)=i^2$, les résultats étant fournis à l'écran

- **On utilisera l'allocation dynamique, mais pas de procédures**

Exercices

Suite de Fibonacci

Ecrire un programme qui détermine l'ensemble des valeurs u_1, u_2, …, u_n (n 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 u_1 u_2 ». 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) »

- **On utilisera l'allocation dynamique et des procédures**