Département MFEE - *Mécanique des Fluides, Energétique et Environnement*

Programmation impérative : méthode de programmation

Thomas Bonometti

Septembre 2021

INP-ENSEEIHT, Ecole Nationale Supérieure d'Electronique, d'Electrotechnique, d'Informatique, d'Hydraulique et des Télécommuniations de Toulouse, 2, rue Charles CAMICHEL, B.P. 7122, 31071 TOULOUSE Cedex 7, France, Tél. +33 (0)5 34 322 000, Fax +33 (0)5 34 322 120

Programmation impérative:
Méthode de Programmation

Sources (d'inspiration forte…) : A la bibliothèque:
Livre 'Programmer en Fortran 90', C. Delannoy, éditions Eyrolles
Et aussi:
Cours N7 d'algorithmique et programmation, Denis Dartus
Polycopié N7 'Algorithmique', M. Alquier & D. Dartus
Cours N7 d'algorithmique et Programmation Impérative 3EA

Plan du cours

Moyens

COURS

5 parties
- Algorithmique + Fortran élémentaire
- Tableaux + Fichiers
- Fonctions + Subroutines
- Allocation dynamique
- Structures de données
- Makefile + Modules + Formats

1 support
- Le langage FORTRAN
- Environnement Linux

HUMAIN

- Une équipe d'enseignants

MATERIEL

- ~ 12 salles en libre service
- ~ 200 calculateurs

EVALUATION

- 1 contrôle individuel
- 1 bureau d'étude en binôme

DEROULEMENT

- Voir planning

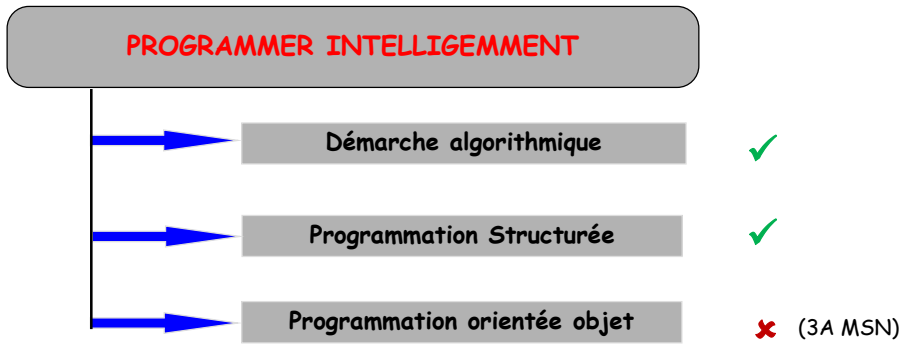
Objectifs Généraux

Quiz

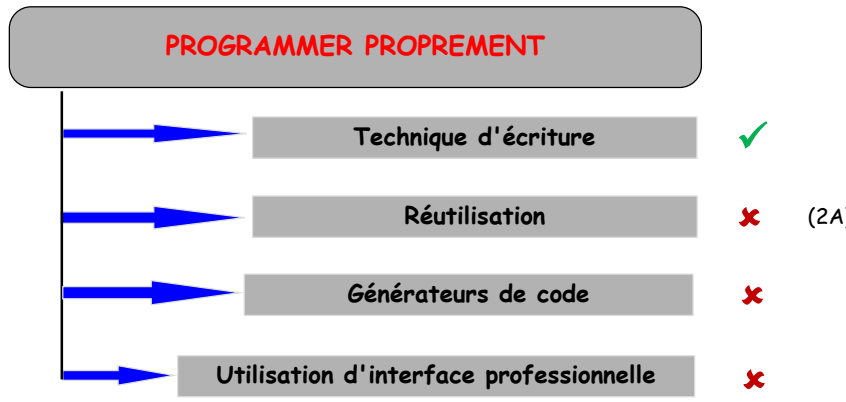
- Physicien utilisateur de l'outil informatique
- ~~Expert en machine, système d'exploitation, langage de programmation, ...~~
- Physicien capable d'analyser puis de résoudre un problème
- ~~Technicien utilisateur d'un code~~
- Utilisateur/développeur de code calcul numérique
- ~~Analyste numérique~~

PROFESSIONNEL de L'INFORMATIQUE du PHYSICIEN

Méthode



Méthode



Physicien - Informaticien

connaissance profonde du domaine d'application

interprétation des résultats

intégration dans des solutions complexes

INP ENSEEIHT TOULOUSE

Programmation impérative: Méthode de Programmation

1. Algorithmique + Fortran élémentaire

Thomas Bonometti

Table des matières

Définition d'un programme

❑ Définition

Un programme est une suite finie d'instructions pré-déterminées destinées à être exécutées de manière automatique par un processeur en vue d'effectuer des traitements, impliquant généralement une interaction avec son environnement.

Exemples d'environnement

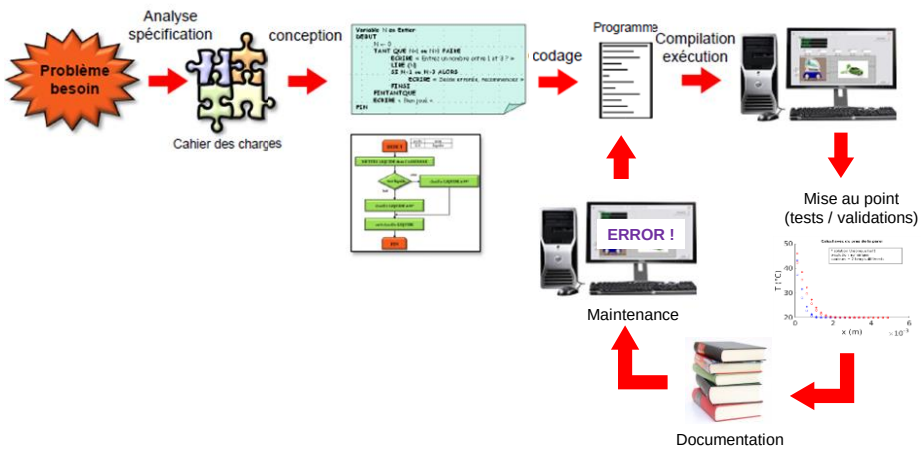
- ❑ Utilisateur humain : traitement de texte, SMS..
- ❑ Un autre système informatique : navigateur internet, guichet automatique bancaire, réseau, etc.
- ❑ Des éléments physiques : capteurs et actionneurs (ABS, régulateur vitesse).

Programme Impératif

- ❑ Programme constitué d'une suite d'ordres (actions) exécutés par un ordinateur.

Il existe d'autres types de programmes qui ne sont pas impératifs.

Formalisation d'un programme



Un programme

Objectif d'un programme :
Obtenir des **Résultats**

Environnement initial :
Constitué par des **Données**

Transformation :
Répond à une **Fonctionnalité**

Définition d'un algorithme

Un algorithme est la description de la suite des actions qu'il faut faire réaliser à un processeur déterminé pour qu'il délivre l'ensemble des résultats à partir de l'ensemble des données

- Indépendant du langage de programmation
- Compréhensible par tous
- Contient tout ce qu'il faut pour réaliser le codage
- Description par le langage pseudo-algorithmique / organigramme

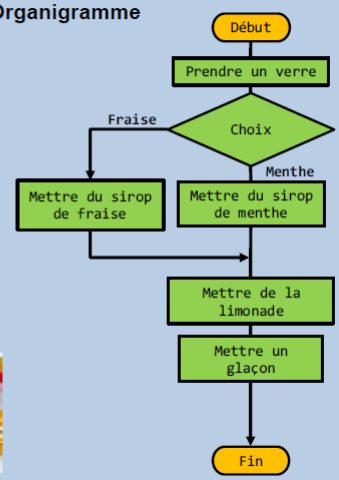
Description d'un algorithme

❑ **Exemple : Algorithme pour un diabolo (menthe ou fraise)**

Pseudo-code

```
Prendre un verre
Si CHOIX menthe
  Alors Mettre du sirop de menthe
Sinon Si CHOIX fraise
  Alors Mettre du sirop de fraise
FinSi
Mettre de la limonade
Mettre un glaçon
```

Organigramme



© cours 3EA

Principe du raffinage

❑ **Pour aider un concepteur à concevoir un algorithme, nous proposons de suivre une démarche à étapes successives permettant d'aider à obtenir une solution en réponse à un cahier des charges.**

1. **Il faut exprimer l'action principale R0 que l'on veut réaliser et avec quel processeur on souhaite le faire**
2. **Il faut identifier les principales données manipulées. Des données supplémentaires pourront être rajoutées plus tard**
3. **Il faut ensuite décomposer l'action principale à l'aide d'actions plus simples notées Ri**

Tant que l'on considère que les actions sont trop complexes pour être comprise par le processeur, la décomposition continue par ajout de niveaux (R1, R2, R3,…)

c'est le processus de raffinage

© cours 3EA

Ex.: préparation du diabolo menthe

1 **Processeur :** Humain

2 **Données :**
- Verre
- Limonade
- Sirop de menthe
- Glaçons

Version Ecrite

3

```
R0 : Préparer un diabolo menthe

R1 : Raffinage de {Préparer un diabolo menthe}
   | Prendre un verre
   | Mettre de la limonade dans le verre
   | Mettre du sirop dans le verre
   | Mettre un glaçon dans le verre

R2 : Raffinage de {Mettre de la limonade dans le verre}
   | Ouvrir le frigo
   | Prendre la limonade
   | Verser la limonade dans le verre
   | Remettre la limonade au frigo
   | Fermer le frigo
```

Trop complexe donc raffinage

Niveau 0 / Niveau 1 / Niveau 2

© cours 3EA

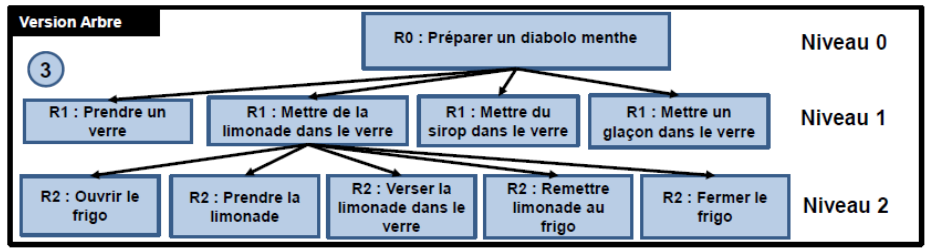
Ex.: préparation du diabolo menthe

1 **Processeur :** Humain

2 **Données :**
- Verre
- Limonade
- Sirop de menthe
- Glaçons

Version Arbre

3



❑ **Le critère d'arrêt** de la décomposition dépend du problème et de l'expérience du concepteur. Il s'agit en général d'obtenir des sous-actions correspondant à des actions élémentaires facilement réalisables par un processeur

© cours 3EA

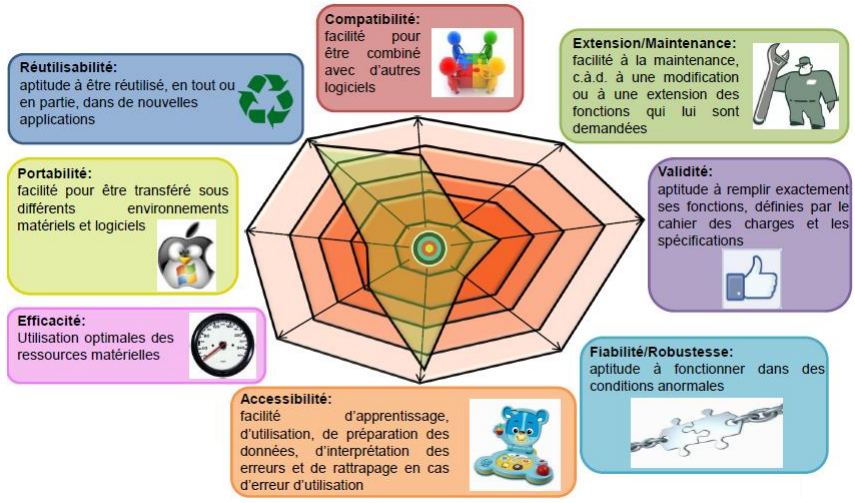
Etape: Programmation

Quelques types de langages

- Les langages machine : définissent le jeu d'instructions élémentaires correspondant aux capacités d'un processeur
- Les langages assembleur : gèrent les adresses logiques étiquettes, déchargent le programmeur du positionnement du programme en mémoire
- **Les langages structurés** : s'appuient sur les structures de contrôle (conditionnelle, répétition) pour éviter la profusion de branchements (et les programmes spaghettis). Structuration en sous-programmes et modules. Exemples : Fortran (1962), Pascal (1969), C (1972), Ada (1983), etc.
- Les langages objets (années 80) : regrouper les données et les traitements. Exemples : SmallTalk (1980), C++ (1983), Java (1995), Python (1990) , Fortran 90,…
- Les langages dédiés (DSL : Domain specific langage) : dédiés à des technologies spécifiques, contrairement aux langages généralistes (GSL : General purpose langage). Exemples : Matlab (1TR), VHDL, JavaScript, etc.

© cours 3EA

Critères de qualité d'un programme



© cours 3EA

n ! en python, en C, en Fortran

```python
def factorielle (n) :
 ######################################
 # fonction factorielle
 # calcule la factorielle d'un entier >= 0
 # paramètre n, n entier
 # précondition n >= 0
 ######################################
       resultat = 1
       for i in range (2, n + 1) :
                resultat = resultat * i
       return resultat


print( factorielle (3))
```

```c
#include<stdio.h>


int appel_factorielle(int n)
{
/* fonction factorielle
   calcule la factorielle d'un entier >=0
   parametre n, n entier donne
   precondition n>=0
*/
   int i,resultat=1;

   for (i=1;i<n+1;i++)  resultat=resultat*i;
   return resultat;
}


void main()
{
int fact;
fact=appel_factorielle(4);
printf("\n 4!=%d \n",fact);
};
```

```fortran
program main
  implicit none
  integer :: factorielle
  print*, factorielle(4)
end program main

function factorielle (n)
  ! calcule la factorielle de n (entier >=0)
  implicit none
  integer, intent(in) :: n
  integer :: i, factorielle
  factorielle = 1
  do i=1,n
     factorielle = factorielle * i
  end do
end function factorielle
```

Attention:
Python langage interprété
C / Fortran langage compilé

Langage interprété vs compilé

Compilateur

Programme qui transforme un code source écrit dans un langage de programmation en un autre langage informatique (langage cible)

- Passage d'un langage compréhensible par l'humain en langage machine (ie C/Fortran, Ada, …)

Interpréteur

Outil d'éxecution d'un langage informatique. Il ne produit pas de code executable, la transformation se fait à la volée et est exécutée ligne après ligne (ce qui ralenti généralement l'exécution)

- Exemple : matlab/Python

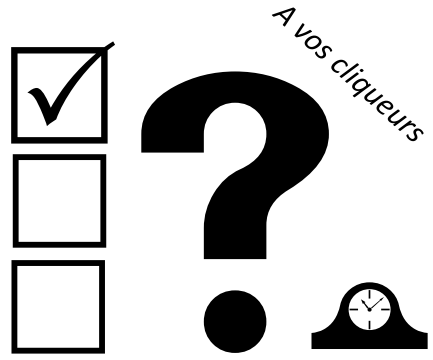
Séquentialité d'un algorithme…

Mon_algo_21

```
i ← 1
j ← 2
i ← i + j
j ← i - j
i ← i - j
```

Mon_algo_1-1

```
i ← 1
j ← 2
j ← i - j
i ← i + j
i ← i - j
```

Séquentialité d'un algorithme… Quiz

Mon_algo_21

```
i ← 1
j ← 2
i ← i + j
j ← i - j
i ← i - j
```

Mon_algo_1-1

```
i ← 1
j ← 2
j ← i - j
i ← i + j
i ← i - j
```

Que valent i et j ?

A. i=1 j=-1
B. i=1 j=2
C. i=2 j=1

#QDLE#Q#ABC*#60#

Séquentialité d'un algorithme… Quiz

Mon_algo_21

```
i ← 1
j ← 2
i ← i + j
j ← i - j
i ← i - j
```

Mon_algo_1-1

```
i ← 1
j ← 2
j ← i - j
i ← i + j
i ← i - j
```

Que valent i et j ?

A. i=1 j=-1
B. i=1 j=2
C. i=2 j=1

#QDLE#Q#A*BC#60#

Règles de bonne programmation

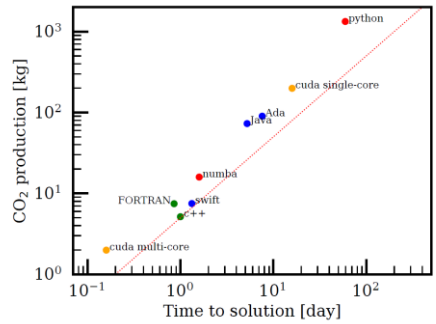
- Utiliser des identificateurs significatifs et commentés pour les noms de constantes, variables, types (identificateurs de types, on en parlera plus tard)
- Toute variable doit être initialisée avant de pouvoir en utiliser la valeur.
- En principe, on donnera une valeur initiale à la variable uniquement au moment ou on en a besoin (et pas nécessairement à la déclaration)
- On choisira, pour un problème donné, les bonnes structures de contrôle (ne pas utiliser un TANT QUE à la place d'un POUR etc.)
- Indenter le code, écrire une seule instruction par ligne
- On vérifiera que les boucles de répétition sont bien écrites, à savoir :
  - La boucle se termine
  - Les variables de la boucle et de la condition ont bien une valeur
  - Le commentaire de fin de boucle sera systématiquement écrit
- Les lectures de données (ou affichages de résultats) doivent être conviviales et fiables

© cours 3EA

FORTRAN = FORmula TRANslator

Pourquoi du Fortran ?

- Langage utilisé dans 80% des logiciels de calcul scientifique (HPC)
- Langage structuré (rigueur de programmation)
- Langage « écolo »? (cf Ref [1])



Ref[1]: Zwart, The ecological impact of high-performance computing in astrophysics, Nature Astronomy, 4(9), 819-822 (2020)

FORTRAN = FORmula TRANslator

1946-48 : Instructions machines fastidieuses et peu fiables
~1950 : Assembleur
1954 : John Backus (IBM) créé le langage symbolique « Mathematical Formula Translating System », dit FORTRAN.
1956 : Premier manuel Fortran I (noms de variables, instruction, FORMAT…)
1957 : Fortran II, premiers compilateurs commerciaux (sous-programmes et fonctions compilables)
1958 : Fortran III est disponible, mais reste interne à la compagnie IBM.
1962 : Fortran IV : langage informatique des scientifiques pendant seize ans.
1966 : Fortran 66 : Création de la norme ANSI.
1978 : Fortran 77 : fin des cartes perforées début de programmation structurée
1991 : Fortran 90 : changement profond du langage: modularité, calcul vectoriel, contrôle de la précision numérique, blocs interfaces, orientation POO
1995 : Fortran 95 : caractérise fonction obsolète
2008 : Fortran 2008 : supporte programmation parrallèle
2018 : Fortran 2018 : Corrections mineures

Corps du programme

Algorithmique

/* déclarations */

Identificateur	Type	Signification

Programme principal
début

{ instructions }

fin

FORTRAN

```
PROGRAM nom_du_programme

IMPLICIT NONE

! déclarations

! instructions

END PROGRAM nom_du_programme
```

Propriétés

- Une variable doit être déclarée : nom, type et rôle (sémantique)
- On accède à une variable par son nom (identificateur)
- A la déclaration, sa valeur est indéterminée
- La valeur d'une variable est une information dynamique. Elle n'est connue qu'à l'exécution du programme.
- La valeur d'une variable est initialisée/modifiée (par instruction d'affectation)

© cours 3EA

FORTRAN

Entier

6 bits : ± 32
16 bits / 2 octets: ± 32768
32 bits / 4 octets: ± 2×10^9 (6 chif. signif.)
64 bits / 8 octets: ± 10^{19} (15 c.s.)

nb d'octets

```
INTEGER :: i, j
INTEGER (KIND=8) :: k, l
! Ex.: +4012  4012  -123
```

Décimal

32 bits / 4 octets: ± 10^{-38} à 10^{38} (6 c.s.)
64 bits / 8 octets: ± 2×10^{-308} à 2×10^{308} (15 c.s.)

```
REAL :: x,y
REAL (KIND=8) :: dx,dy
! Ex. : 12.43      -0.38      -.38        4.     .27
!        12.43E0   1.243E1   1.243e+1
```

Logique

```
LOGICAL :: etu
! Ex.:  .FALSE.   .TRUE.
```

Alphabétique (au sens large)

nb de caractères

```
CHARACTER (len=28) :: nom
! Ex.: 'toto 1'     'je sais qu on ne sait jamais'
```

Ecriture / Lecture à l'écran

Algorithmique	FORTRAN
	INTEGER :: a, i CHARACTER (len=20) :: phrase
Ecriture	
Ecrire(a)	WRITE(*,*) a ⟺ PRINT*, a
Ecrire(a,i)	WRITE(*,*) a, i
Ecrire(« Le resultat est: »,a)	WRITE(*,*) "Le resultat est: " , a
	raccourci d'écriture
Lecture	
Lire(a)	READ(*,*) a ⟺ READ*, a
Lire(a,i)	READ(*,*) a, i
	raccourci d'écriture

Exemple de la facture

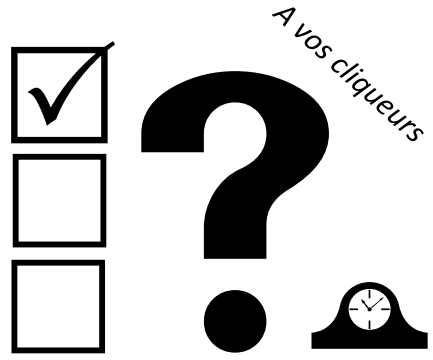
Algorithmique

Facture

Lire (p1)
Lire (n1)
pht1 ← n1 * p1
Lire (p2)
Lire (n2)
pht2 ← n2 * p2
pht ← pht1 + pht2
pttc ← pht * 1.186
Ecrire (pttc)

FORTRAN

```
PROGRAM FACTURE
   IMPLICIT NONE

   INTEGER :: n1, n2
   REAL :: p1, p2, pht1, pht2, pht, pttc

   WRITE(*,*) "Prix HT unitaire du 1er type d'articles ?"
   READ(*,*) p1
   WRITE(*,*) "Nombre d'articles du 1er type ?"
   READ(*,*) n1
   pht1 = real(n1) * p1
   WRITE(*,*) "Prix HT unitaire du 2eme type d'articles ?"
   READ(*,*) p2
   WRITE(*,*) "Nombre d'articles du 2eme type ?"
   READ(*,*) n2
   pht2 = real(n2) * p2
   pht = pht1 + pht2
   pttc = pht * 1.186
   WRITE(*,*) "La facture est de: ", pttc, " euros"

END PROGRAM FACTURE
```



Quel programme est correct?

Quiz

A.

```
program TOTO

implicit none

integer :: a

write(*,*) "Donner la valeur de a (entier)"
read(*,*) a

print*, "La valeur de a est", a

end program TOTO
```

B.

```
program TATA

implicit none

integer :: a

print*, 'Donner la valeur de a (entier)'
read*, a
write*, "La valeur de a est", a

end program TATA
```

#QDLE#Q#A*B#60#

Opérateurs arithmétiques 1/2

Algorithmique	FORTRAN
addition, soustrac.°, multipl.°, division	+ - * /
	! Ex.: 7/5 donne la valeur 1
	! 7./5. donne la valeur 1.4
Attention: Faire des opérations entre objets de même type uniquement !	! 7./5 donne une valeur fonction du compilateur…
moins (unaire)	- ! Ex.: -3 -1.2 -98.e-2

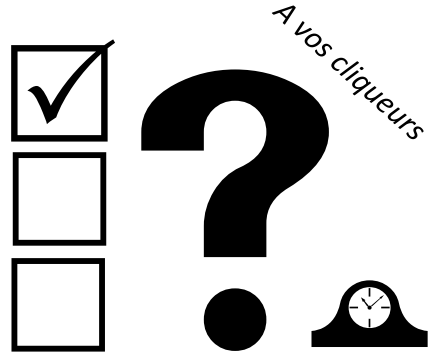
Opérateurs arithmétiques 2/2

Algorithmique	FORTRAN
élévation à la puissance	**
	! Ex.: 2**3 donne 8 (entier)
	! 2**(-3) donne 0 (entier)
	! 2.**3
	! 2**3. donnent 8. (réel)
	! 2.**3.
	! 2.**(-3)
	! 2**(-3.) donnent 0.125 (réel)
	! 2.**(-3.)
	! -2**3 donne -8 (entier)
	! (-2.)**3 donne -8. (réel)
Remarque: $e^{3\ \ln(-2.)}$ n'est pas possible	! (-2.)**3. à éviter

Affectation

Algorithmique	FORTRAN
a ← b	a = b
i ← i+1	i = i+1
En programmation a = a + b est tout-à-fait possible même avec a ≠ 0 et b ≠ 0 …	a = a + b

Attention: Faire des affectations entre objets de même type uniquement !



Qu'affiche ce programme ?

Quiz

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + n/p
  q = p + x*n
  PRINT*, p + x*n
  PRINT*, q

END PROGRAM TOTO
```

	A.	B.	C.	D.
`PRINT*, x + n/p`	1.8	1.8	1.2	1.2
`PRINT*, p + x*n`	8.6	8	8.6	8.6
`PRINT*, q`	8	8	8	8.6

#QDLE#Q#ABC*D#60#

Problème de transtypage

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + n/p
  q = p + x*n
  PRINT*, p + x*n
  PRINT*, q

END PROGRAM TOTO
```

réel + entier/entier !!!

entier = entier + réel*entier !!!

entier + réel*entier !!!

C.

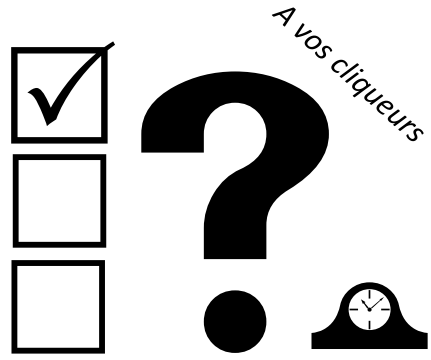
1.2

8.6

8

Solution au problème de transtypage

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + real(n)/real(p)
  q = p + int(x)*n
  PRINT*, p + int(x)*n
  PRINT*, q

END PROGRAM TOTO
```

Solution au problème de transtypage

```
PROGRAM TOTO
IMPLICIT NONE
INTEGER :: n, p, q
REAL :: x

  n = 3
  p = 5
  x = 1.2
  PRINT*, x + real(n)/real(p)
  q = p + int(x)*n
  PRINT*, p + int(x)*n
  PRINT*, q

END PROGRAM TOTO
```

Quiz

	A.	B.	C.	D.
	1.8	1.8	1.2	1.2
	8.6	8	8.6	8.6
	8	8	8	8.6

#QDLE#Q#AB*CD#60#

Expressions logiques

Algorithmique	FORTRAN
Comparaison d'entiers/réels	== ! égal à /= ! différent de < <= > >=
Comparaison de logiques	.EQV. .NEQV.
Attention #1: Comparaisons entre objets de même type uniquement !	! Ex.: n == m (n, m entiers) ! Ex.: x == y (x, y réels)
Attention #2: tests d'égalité entre réels à éviter absolument !	 ! 2 réels ne sont égaux qu'aux erreurs de troncature près…
Connecteurs logiques: et / ou	.AND. .OR.

Traitement conditionnel 1/5

Algorithmique

Si (condition)
Alors

FORTRAN

```
IF (expression_logique) THEN

  instruction 1
  instruction 2
  …
  instruction n

END IF
```


Traitement conditionnel 2/5

Algorithmique

si (condition)
Alors

Sinon

FORTRAN

```
IF (expression_logique) THEN

  instruction 1
  instruction 2
  …
  instruction n

ELSE

  instruction 1
  instruction 2
  …
  instruction n

END IF
```

Traitement conditionnel 3/5

Algorithmique	FORTRAN

Si (condition 1)
Alors
|
Sinon si (condition 2)
Alors
|
Sinon
|
	IF (expression_logique) THEN
	instruction 1
	…
	instruction n
	ELSE IF (expression_logique) THEN
	instruction 1
	…
	instruction n
	ELSE
	instruction 1
	…
	instruction n
	END IF

Traitement conditionnel 4/5

Algorithmique	FORTRAN

Version classique	IF (expression_logique) THEN
	instruction simple
	END IF

Raccourci d'écriture (quand instruction simple uniquement)	IF (expression_logique) instruction_simple

	IF (a==0) THEN
	WRITE(*,*) "a est egal a 0"
	END IF

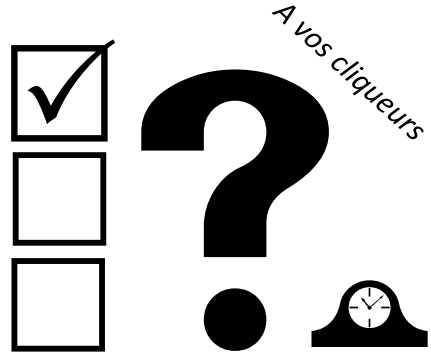
	IF (a==0) WRITE(*,*) "a est egal a 0"

Algorithmique

Variante
utilisant la commande
SELECT CASE

FORTRAN

```
SELECT CASE (n)
  CASE (0)                  ! n = 0
    print *, 'eliminatoire'
  CASE (5,6,7,8,9,10)       ! n = 5 à 10
    print *, 'mention passable'
  CASE (11:15)              ! n = 11 à 15
    print *, 'mention bien'
  CASE (16:20)              ! n = 16 à 20
    print *, 'mention tres bien'
  CASE DEFAULT              ! autres cas
    print *, 'pas de mention'
END SELECT
```

Qu'affiche ce programme à la fin si a='oui' b='non' c='oui' age=16 ?

Quiz n°1

```
PROGRAM RESTO
IMPLICIT NONE

 CHARACTER (LEN=3) :: a, b, c
 INTEGER :: age

PRINT*, 'Voulez boire (oui/non)?'
READ*, a

IF (a == 'oui') THEN
   PRINT*, 'Une boisson alcolisee (oui/non)?
   READ*, b
   IF (b == 'oui') THEN
        PRINT*, 'Donner votre age'
        READ*, age
    END IF
END IF

PRINT*, 'Voulez-vous manger (oui/non)?'
READ*, c

IF (b == 'oui' .AND. age < 18) THEN
    PRINT*, 'Boire avant 18 ans est interdit'
ELSE IF (a == 'oui' .AND. c == 'oui') THEN
    PRINT*, 'Nous avons des menus'
ELSE IF (a == 'oui' .OR. c == 'oui') THEN
    PRINT*, 'Bon appetit'
ELSE
    PRINT*, 'Au revoir'
END IF

END PROGRAM RESTO
```

A. 'Boire avant 18 ans est interdit'

B. 'Nous avons des menus'

C. 'Bon appetit'

D. 'Au revoir'

E. 'Nous avons des menus'
'Bon appetit'

#QDLE#Q#AB*CDE#60#

Qu'affiche ce programme si l'utilisateur entre 1. et 1. ?

Quiz n°2

```
program egalite

implicit none

real :: x,y

print*,"Donner deux valeurs réelles"
read*, x,y

if (x==y) then
    print*,"Les deux valeurs sont égales"
else
    print*,"Les deux valeurs sont différentes"
end if

end program egalite
```

A. 'Les deux valeurs sont différentes'

B. 'Les deux valeurs sont égales'

C. On ne sait pas

#QDLE#Q#ABC*#30#

Solution au problème de test d'égalité sur des réels

```
program inegalite

implicit none

real :: x,y

print*,"Donner deux valeurs réelles"
read*, x,y

if (abs(x-y)<=1.e-8) then
    print*,"Les deux valeurs sont égales à 10^-8 près"
else
    print*,"Les deux valeurs sont différentes"
end if

end program inegalite
```


Traitement itératif 1/3

Algorithmique	FORTRAN
Si nombre d'itérations CONNU	INTEGER :: i, n
Pour i de 1 à n faire	DO i = 1, n
|	instruction 1 … instruction k
	END DO

Traitement itératif 2/3

Algorithmique

Variante
(boucle à progression décroissante)

Pour i **de** n **à** 1 **faire**

à utiliser avec précautions…

FORTRAN

```
INTEGER :: i, n
DO i=n,1,-1
  instruction 1
  …
  instruction k
END DO
```

Cas général

```
INTEGER :: i, nd, na, pas
DO i=nd,na,pas
  instruction 1
  …
  instruction k
END DO
```


Traitement itératif 3/3

Algorithmique

Si nombre d'itérations INCONNU

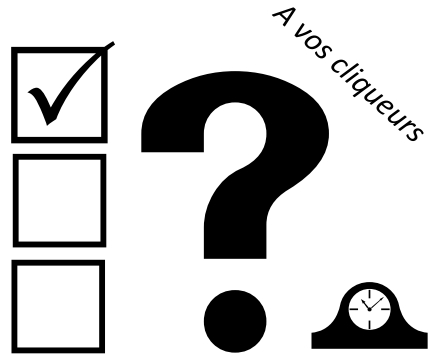
Tant que (condition) **faire**

répété tant que la condition est « vraie »

Attention aux boucles infinies…

FORTRAN

```
DO WHILE (expression_logique)

  instruction 1
  …
  instruction n

END DO
```



Qu'affiche ce programme ?

Quiz n°1

```
program boucle

implicit none

integer :: i

do i=1,3
   print*,"Coucou"
end do

end program boucle
```

A.	B.	C.	D.
Coucou Coucou	Coucou Coucou Coucou	Coucou Coucou Coucou Coucou	Ni A, ni B, ni C

#QDLE#Q#AB*CD#60#

Qu'affiche ce programme ?

Quiz n°2

```
program boucle_bis

implicit none

integer :: i

i=0
do while (i<=3)
   print*,"Coucou"
   i=i+1
end do

end program boucle_bis
```

A.	B.	C.	D.
Coucou Coucou	Coucou Coucou Coucou	Coucou Coucou Coucou Coucou	Ni A, ni B, ni C

#QDLE#Q#ABC*D#60#

Exercice

Suite de Fibonacci

Ecrire un programme en FORTRAN qui détermine la énième valeur 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)

IMPLICIT NONE, pour quoi faire ?

Si une variable commençant par i, j, k, l, m, n est non-déclarée
→ FORTRAN affecte implicitement le type INTEGER
Pour les autres variables non-déclarées
→ FORTRAN affecte implicitement le type REAL

Exemple sans IMPLICIT NONE

```
PROGRAM TOTO
  INTEGER :: k, nbre
  nbre = 5
  k = nbr + 1
  PRINT*, k
END PROGRAM TOTO
```

- Pas d'erreur à la compilation
- 1er lancement: « 1105920329 »
- 2nd lancement: « -459992759 »
- 3ème lancement: « -1909489335 »

Exemple avec IMPLICIT NONE

```
PROGRAM TOTO
  IMPLICIT NONE
  INTEGER :: k, nbre
  nbre = 5
  k = nbr + 1
  PRINT*, k
END PROGRAM TOTO
```

- Message d'erreur à la compilation

MAJUSCULES et minuscules en Fortran…

Le fortran ne fait PAS DE DISTINCTION entre majuscules et minuscules !!!

Toto	=	TOTo	=	TotO	=	toTo	=
TOTO	=	TOtO	=	tOTo	=	totO	=
Toto	=	ToTO	=	tOtO	=	Toto	=
tOto	=	tOTO	=	toTO	=	ToTo	

Mise en œuvre d'un algorithme

Comment UTILISER un programme lorsqu'il est écrit?

- Il faut l'EXECUTER en créant un EXECUTABLE obtenu en COMPILANT le code source avec un COMPILATEUR (libre ou commercial)
- Le code source doit se trouver dans un fichier dont l'extension est « .f90 »

Qu'est-ce la compilation?

1° Etape de création de fichiers objets correspondant aux programmes. Le code source est analysé, et s'il est correct, un ensemble d'instructions processeur correspondant aux instructions du code source est généré dans un fichier objet (d'extension *.o).

2° Etape d'édition de liens liant les instructions desfichiers objets aux librairies correspondantes afin de créer le fichier exécutable (car on utilise des fonctions qui sont préprogrammées: lire/écrire un caractère, fonctions mathématiques, etc, et qui sont regroupées dans des librairies).

Compilation (ligne de commandes)

FORTRAN

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

! Compilation (version très courte)
> ls
> prog.f90
> gfortran prog.f90
> ls
> prog.f90   a.out
```

FORTRAN

```
! Compilation (version courte)
> ls
> prog.f90
> gfortran prog.f90 –o coucou.exe
> ls
> prog.f90   coucou.exe

! Compilation (version longue)
> ls
> prog.f90
> gfortran prog.f90 –c
> gfortran prog.o –o coucou.exe
> ls
> prog.f90   prog.o   coucou.exe
```


Mise en œuvre d'un algorithme

Programmation

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

Compilation (lignes de commandes)

```
! Compilation (ligne de commandes)
> ls
> prog.f90
> gfortran prog.f90 –o coucou.exe
> ls
> prog.f90   coucou.exe
```

Exécution

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

INP ENSEEIHT

Programmation impérative: Méthode de Programmation

2. Tableaux + Fichiers

Thomas Bonometti

Table des matières

Pourquoi un tableau ?

Tableau:

Ensemble ordonné d'éléments de même type désigné par un identificateur unique

Pourquoi un tableau? Parce que ça…

- a une signification physique propre
  Ex.: projection du vecteur sur l'axe i
- a le sens de la numérotation
  Ex.: i-ème valeur
- sert de lien entre les tableaux
  Ex.: a(i,j)*x(j)

Les tableaux simples (de rang 1)

Algorithmique	FORTRAN
	! définition et déclaration de variables REAL, DIMENSION(100) :: v1 INTEGER, DIMENSION(100) :: u1 LOGICAL, DIMENSION(100) :: w1
Soit un vecteur de 100 réels Soit un vecteur de 100 entiers Soit un vecteur de 100 logiques	
	! affectation et accès à un champ
Le vecteur v1 est nul	DO i = 1,100 v1(i) = 0. ⟺ v1(:)=0. ⟺ v1=0. (ambigu) END DO à éviter
La 50ème composante de v1 reçoit 1 La 12ème composante de v1 reçoit la 50ème	v1(50) = 1. v1(12) = v1(50)

Manipulation des tableaux

Algorithmique	FORTRAN
	! définition et déclaration de variables
Soient 2 vecteurs de 100 réels	REAL, DIMENSION(100) :: v1, v2
	! Manipulations d'un tableau
La 50ème composante de v2 reçoit la 32ème de v1	v2(50) = v1(32)
v2 ← v1	DO i = 1,100 v2(i) = v1(i) ⟺ v2(:) = v1(:) ⟺ v2 = v1 (ambigu) END DO 
Chaque composante de v2 reçoit le double de la composante de v1 plus 3	v2=2.*v1+3.
Chaque composante de v1 reçoit le produit des composantes de v1 et v2	v1=v1*v2  (ambigu)

Bornes des indices

Algorithmique	FORTRAN
Créer 1 vecteur comportant 4 éléments de type réels	REAL, DIMENSION(4) :: v1 ! → v(1), v(2), v(3), v(4)
Créer 1 vecteur comportant 4 éléments de type réels dont l'indice commence par 1	REAL, DIMENSION(1:4) :: v1 ! → v(1), v(2), v(3), v(4)
Créer 1 vecteur comportant 4 éléments de type réels dont l'indice commence par 0	REAL, DIMENSION(0:3) :: v1 ! → v(0), v(1), v(2), v(3)
Créer 1 vecteur comportant 4 éléments de type réels dont l'indice commence par -10	REAL, DIMENSION(-10:-7) :: v1 ! → v(-10), v(-9), v(-8), v(-7)

Problème de dépassement d'indices

Algorithmique

Soient 2 vecteur de 10 réels

Que se passe-t-il pour les instructions suivantes?

Attention aux dépassement de tableaux !!!

Toujours vérifier les expressions en début et fin de boucle

FORTRAN

```
REAL, DIMENSION(10) ::  v1, v2

! v(1) v(2) … v(10) existent
! v(0), v(11), v(99), v(-30) n'existent pas…

v1(15) = 0.
v1(15) = v2(3)
v1(3)   = v2(15)

DO i = 1, 10
   v1(i) = v2(i+1)
END DO
```


Ecriture à l'écran d'un tableau

Algorithmique

Soit u(i)=1.2 avec i=1, …, 100

{ **Ecrire** à l'écran u en colonne }
  Ex.: 1.2
       1.2
       …
       1.2

{ **Ecrire** à l'écran u sur une ligne }
  Ex.: 1.2 1.2 … 1.2

FORTRAN

```
INTEGER :: i
REAL, DIMENSION (100) :: u
u(:) = 1.2

DO i = 1, 100
   WRITE(*,*) u(i)
END DO
```

retour à la ligne à chaque appel de l'instruction

```
WRITE(*,*) ( u(i), i=1,100 )
```

WRITE(*,*) u

Exemple du tri (2)

Tri

Fonctionnalité : On veut trier par ordre croissant les éléments d'un vecteur contenant n éléments (n<=100)

1° Environnement

Identificateur	Type	Signification
n	Entier	nombre d'éléments à trier
u(1), … , u(n)	Réel	vecteur de départ
v(1), … , v(n)	Réel	vecteur « trié »

Exemple du tri (2)

2° Traitement

```
Tri
|  Lire (n)
|  Lire (u(1), …, u(n))
|
|  ! Détermination de la valeur « umax »
|  !  des u(1), … u(n)
|  umax ← u(1)
|  Pour i de 2 à n faire
|  |  Si  u(i) >= umax
|  |     Alors
|  |     |  umax ← u(i)
```

```
! Recherche de la plus petite valeur umin
!  de u et de son emplacement jmin
Pour i de 1 à n faire
|  jmin ← 1
|  umin ← umax
|  Pour j de 1 à n faire
|  |  Si  u(j) <= umin
|  |     Alors
|  |     |  umin ← u(j)
|  |     |  jmin ← j
|  ! On met la valeur min dans v
|  v(i) ← umin
|  ! On « écrase » la plus petite valeur de u
|  u(jmin) ← umax

Affiche (v(1),…, v(n))
```

Exemple du tri (2)

```
PROGRAM TRI

   IMPLICIT NONE

   ! Declarations
   INTEGER :: n, i, j, jmin
   REAL :: umax, umin
   REAL, DIMENSION(100) :: u, v

   ! Instructions

   WRITE(*,*)"Donner le nb d'elements a trier"
   READ(*,*) n
   DO i = 1,n
      WRITE(*,*)'Valeur du',i,'eme element?'
      READ(*,*) u(i)
   END DO

   umax = u(1)
   DO i = 2, n
       IF( u(i) >= umax ) umax = u(i)
   END DO
```

```
   DO i = 1, n
       jmin = 1
       umin = umax
       DO j = 1, n
            IF ( u(j) <= umin ) THEN
                  umin = u(j)
                  jmin = j
             END IF
        END DO
        v(i) = umin
        u(jmin) = umax
   END DO

   WRITE(*,*) 'Les element tries sont:'
   WRITE(*,*) ( v(i), i = 1, n )

END PROGRAM TRI
```


Les matrices (tableaux de rang 2)

Algorithmique	FORTRAN
	! définition et déclaration
Soit une matrice de 10×10 réels Soit une matrice de 10×10 entiers	REAL, DIMENSION(10,10) :: m1 INTEGER, DIMENSION(10,10) :: m2
! Ex. : Création de la matrice Identité	! affectation et accès à un champ

1° création d'une matrice nulle	DO i = 1,10
	  DO j = 1, 10
	  m1(i,j) = 0. ⟺ m1(:,:)=0.
	  END DO
	END DO

2° remplissage de la diagonale de 1	DO i = 1,10 m1(i,i) = 1. END DO

Ecriture à l'écran d'une matrice

Algorithmique	FORTRAN
	! déclaration REAL, DIMENSION(10,10) :: M INTEGER :: i,j
Soit M la matrice identité de 10×10 réels	! instructions M(:,:)=0. DO i = 1,10 M(i,i) = 1. END DO
{ **Ecrire** à l'écran M }	PRINT*, 'M=' DO i = 1,10 PRINT*, (M(i,j), j=1,10) END DO
M= 1. 0. 0. 0. … 0. 1. 0. 0. … 0. 0. 1. 0. … …	

Rang, étendue et profil des tableaux

Etendue : nb d'éléments suivant une dimension

Rang : nb de dimensions (R≤7 en Fortran 90)

REAL, DIMENSION (n1, n2, …, nR) :: t

Profil : liste des étendues

! Exemples

```
INTEGER, DIMENSION (5) :: t                  ! Rang = 1 ;  Profil = (5)  ;  Etendue = 5
INTEGER, DIMENSION (10, 5) :: t1             ! Rang = 2 ;  Profil = (10,5)  ;  Etendue = 10,5
REAL, DIMENSION (-2:7, 0:4) :: t2            ! t2 a le même profil que t1 (même rang et étendue)
LOGICAL, DIMENSION (5, 5, 5, 5) :: test  ! Rang = 4  ;  Profil = (5,5,5,5)  ;  Etendue = 5,5,5,5
```

Manipulation de tableaux

Algorithmique

Exemple 1:
Calcul d'un produit scalaire
x=U.V

FORTRAN

```
REAL, DIMENSION(10) ::  u, v     ! Donnee
REAL ::  x                       ! Resultat
INTEGER :: i

! Produit scalaire
x = 0.
DO i = 1,10
    x = x + u(i)*v(i)
END DO
```


Manipulation de tableaux

Algorithmique

Exemple 2:
Calcul d'un produit matrice vecteur
B=AX

FORTRAN

```
REAL, DIMENSION(10,5) ::  a     ! Donnee
REAL, DIMENSION(5) ::  x        ! Donnee
REAL, DIMENSION(10) ::  b       ! Resultat
INTEGER :: i, j

! Produit matrice-vecteur
DO i = 1,10
  DO j = 1, 5
      b(i) =  a(i,j)*x(j)
  END DO
END DO
```

Suite de Fibonacci

Ecrire un programme en FORTRAN qui détermine l'ensemble des valeurs $u_1, u_2, \ldots, u_n$ ($n \leq 100$ étant fourni en donnée) de la suite définie comme suit:

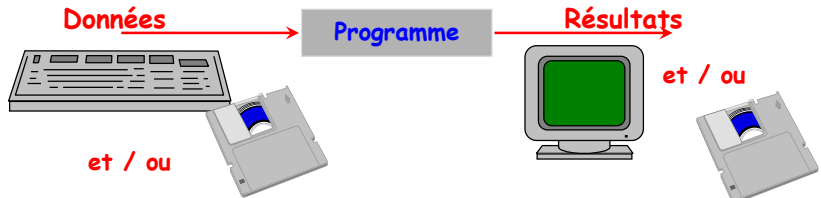
$u_1 = 1$ $u_2 = 1$ $u_n = u_{n-1} + u_{n-2}$ (pour n>2)

Les fichiers

Notion de fichier
Organisation

PROBLEMATIQUE

Pas de conservation des résultats
Entrées des données fastidieuses



stockage sur support magnétique sous forme de fichiers

Organisation d'un fichier

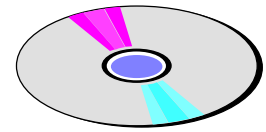
Un fichier est une suite d'articles (ou enregistrements) en général de même nature

Exemples :
- fichier d'adresses
- fichier de composants électroniques
- fichier de bouteilles de vin

Exception : les fichiers textes

Accès séquentiel

Accès direct



Pour relire un fichier que l'on a constitué, il faut connaître la forme de son organisation

Fichiers formatés / non-formatés

2 façons de représenter l'information:

Fichiers formatés (ou « fichiers texte »)

- transfert « mémoire → fichier » avec modification de l'info
  (codage binaire -> codage en base 10 -> chaine de caractère)
- volume des infos dans fichiers > volume des infos dans mémoire ☹
- fichiers lisibles avec éditeurs de texte + portabilité entre machines ☺

Fichiers non-formatés (ou « fichier binaires »)

- transfert « mémoire → fichier » sans modification de l'info
- volume des infos dans fichiers = volume des infos dans mémoire (codage binaire) ☺
- vitesse de transfert plus rapide que pour les fichiers formatés ☺
- fichiers non-lisibles avec éditeurs de texte + pb de portabilité possibles ☹

Exemples de fichiers formatés / non-formatés ouvert avec un éditeur de texte

Fichier formaté

```
62        62        3
0.0000000000000000      0.0000000000000000
0.0000000000000000      0.0000000000000000
0.1000000000000000      0.1500000000000000
0.2500000000000000      0.3000000000000000
0.4000000000000000      0.4500000000000000
0.5500000000000000      0.6000000000000001
```

Fichier non-formaté

```
?¹ZÊ…ý/á? É
àë?úR<…
    Öè?ÈeyGê?~2à8Jñ?RÏ„ûð?òãL!óãó?Qæ´
    LPE÷?ÔßO8çö?•úÛbø?ªÿ;Dh¶ü?y®Bd?@
    ¹- DXi?@¼ð¦=Æ¥?@eÖ¼ZX@ÁÝ
            Ýª@gnB @¥¤ísÛ8 @"ÝÞ«ãÂ
    @D1YÆs@¶¼SÕ          @Q`4ð©
    @ÇÕ%[
```


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)

Lecture/écriture dans un fichier (séquentiel / formaté)

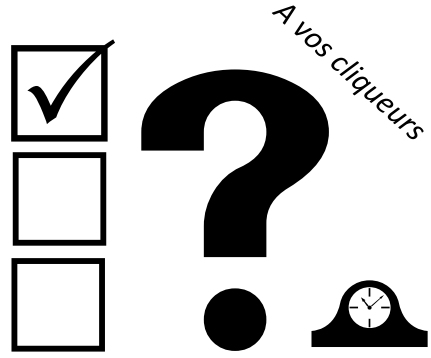
A chaque fichier est associé un numéro (numéro d'unité logique) qui devient le « nom » du fichier pour le programme

Algorithmique

Ouvrir le fichier toto.txt
Si le fichier existe: il l'ouvre
Si le fichier n'existe pas: il le crée
Lire la valeur de x et y dans toto.txt
Ecrire la valeur de z dans toto.txt
Fermer le fichier toto.txt

FORTRAN

```
PROGRAM lecture_fichier
IMPLICIT NONE
REAL :: x,y,z

OPEN (10, file = 'toto.txt')
Z=99.9
READ (10, *) x, y
WRITE (10, *) z
CLOSE (10)
END PROGRAM lecture_fichier
```

Que se passe-t-il à l'exécution du programme avec ce fichier toto.txt ?

Quiz n°1

```
PROGRAM lecture_fichier
IMPLICIT NONE
REAL :: x,y,z

 OPEN(10,file='toto.txt')
 Z=99.9
 READ(10,*) x, y
 WRITE(10,*) z
 CLOSE(10)

END PROGRAM lecture_fichier
```

Contenu du fichier toto.txt :

```
10.2    21.2
```

A. Erreur à l'exécution

B. Exécution OK et fichier modifié

C. Exécution OK et fichier non-modifié

Que se passe-t-il à l'exécution du programme avec ce fichier toto.txt ?

Quiz n°2

```
PROGRAM lecture_fichier
IMPLICIT NONE
REAL :: x,y,z

 OPEN(10,file='toto.txt')
 Z=99.9
 READ(10,*) x, y
 WRITE(10,*) z
 CLOSE(10)

END PROGRAM lecture_fichier
```

Contenu du fichier toto.txt :

```
10.2    21.2    32.3
```

A. Erreur à l'exécution

B. Exécution OK et fichier modifié

C. Exécution OK et fichier non-modifié

Que se passe-t-il à l'exécution du programme avec ce fichier toto.txt ?

Quiz n°3

```
PROGRAM lecture_fichier
IMPLICIT NONE
REAL :: x,y,z

 OPEN(10,file='toto.txt')
 Z=99.9
 READ(10,*) x, y
 WRITE(10,*) z
 CLOSE(10)

END PROGRAM lecture_fichier
```

Contenu du fichier toto.txt :

```
10.2
21.2
```

A. Erreur à l'exécution

B. Exécution OK et fichier modifié

C. Exécution OK et fichier non-modifié

Que se passe-t-il à l'exécution du programme avec ce fichier toto.txt ?

Quiz n°4

```
PROGRAM lecture_fichier
IMPLICIT NONE
REAL :: x,y,z

 OPEN(10,file='toto.txt')
 Z=99.9
 READ(10,*) x, y
 WRITE(10,*) z
 CLOSE(10)

END PROGRAM lecture_fichier
```

Contenu du fichier toto.txt :

```
10.2    21.2
32.3
```

A. Erreur à l'exécution

B. Exécution OK et fichier modifié

C. Exécution OK et fichier non-modifié

Que se passe-t-il à l'exécution du programme avec ce fichier toto.txt ?

Quiz n°5

```
PROGRAM lecture_fichier
IMPLICIT NONE
REAL :: x,y,z

 OPEN(10,file='toto.txt')
 Z=99.9
 READ(10,*) x, y
 WRITE(10,*) z
 CLOSE(10)

END PROGRAM lecture_fichier
```

Contenu du fichier toto.txt :

```
10.2    21.2    ! Valeurs de x et y
```

A. Erreur à l'exécution

B. Exécution OK et fichier modifié

C. Exécution OK et fichier non-modifié

Que font ces deux programmes ?

Quiz n°6

```
PROGRAM ecriture_fichier
IMPLICIT NONE
REAL, DIMENSION(5) :: x
INTEGER :: i

 x(:)=12.34
 OPEN(10,file='res.dat')
 WRITE(10,*) (x(i),i=1,5)
 CLOSE(10)

END PROGRAM ecriture_fichier
```

```
PROGRAM ecriture_fichier
IMPLICIT NONE
REAL, DIMENSION(5) :: x
INTEGER :: i

 x(:)=12.34
 OPEN(10,file='res.dat')
 DO i=1,5
   WRITE(10,*) x(i)
 END DO
 CLOSE(10)

END PROGRAM ecriture_fichier
```

Que font ces deux programmes ?

Quiz n°6

```
PROGRAM ecriture_fichier
IMPLICIT NONE
REAL, DIMENSION(5) :: x
INTEGER :: i

 x(:)=12.34
 OPEN(10,file='res.dat')
 WRITE(10,*) (x(i),i=1,5)
 CLOSE(10)

END PROGRAM ecriture_fichier
```

Contenu du fichier res.dat :

```
12.34  12.34  12.34  12.34  12.34
```

```
PROGRAM ecriture_fichier
IMPLICIT NONE
REAL, DIMENSION(5) :: x
INTEGER :: i

 x(:)=12.34
 OPEN(10,file='res.dat')
 DO i=1,5
   WRITE(10,*) x(i)
 END DO
 CLOSE(10)

END PROGRAM ecriture_fichier
```

Contenu du fichier res.dat :

```
12.34
12.34
12.34
12.34
12.34
```

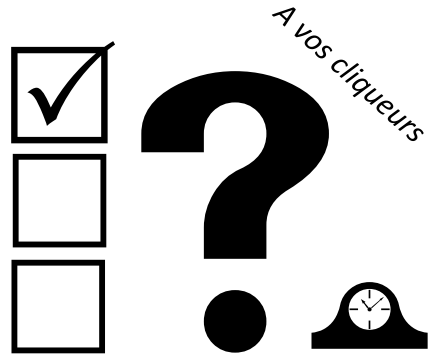

Options d'ouverture d'un fichier

FORTRAN

```
! Version courte
OPEN (10, file = 'toto.txt')
! Version moyenne
OPEN (unit = 10, file = 'toto.txt', form = 'formatted', status = 'new')
```

n° d'unité — nom du fichier — type de représentation — statut du fichier

```
! Version très longue…
OPEN ( unit = 10, file = 'toto.txt',          &
       form = 'formatted'/'unformatted',      &  ! fichier formaté ou non
       status = 'old'/'new',                  &  ! fichier existant ou nouveau
       access = 'sequential'/'direct',        &  ! accès séquentiel ou direct
       position = 'rewind'/'append',          &  ! Position du 'curseur' en début ou en fin
       advance = 'yes'/'no',                  &  ! Revient à la ligne ou non
       iostat = err )                            ! err=0 si tout s'est bien passé, ≠ 0 sinon
```



Ces programmes sont-ils équivalents? Quiz

```
! Version n°1
PROGRAM COUCOU
  IMPLICIT NONE
  OPEN(33, file='coucou.txt')
  WRITE(33,*) 'Coucou'
  CLOSE(33)
END PROGRAM COUCOU
```

```
! Version n°2
PROGRAM COUCOU
  IMPLICIT NONE
  OPEN(22, file='coucou.txt')
  WRITE(*,*) 'Coucou'
  CLOSE(22)
END PROGRAM COUCOU
```

```
! Version n°3
PROGRAM COUCOU
  IMPLICIT NONE
  INTEGER :: num_fich
  CHARACTER (len=20) :: nom_fich
  num_fich = 44
  nom_fich = 'coucou.txt'
  OPEN (unit=num_fich, file = nom_fich, &
            form = 'formatted', status = 'new')
  WRITE(num_fich,*) 'Coucou'
  CLOSE(num_fich)
END PROGRAM COUCOU
```

A. oui

B. Version n°1 ⇔ Version n°2 ≠ Version n°3

C. Version n°1 ⇔ Version n°3 ≠ Version n°2

D. Version n°1 ≠ Version n°2 ⇔ Version n°3

E. ils sont tous différents

Exemple du tri (3)

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 contenant les informations sous la forme suivante:

« n
u(1)
...
u(n) »

On veut écrire le résultats dans un fichier texte appelé res.dat ayant la même forme que don.dat (mais avec les éléments rangés dans le bon ordre…).

Exemple du tri (3)

```
PROGRAM TRI
  IMPLICIT NONE
  ! Declarations
  INTEGER :: n, i, j, jmin
  REAL :: umax, umin
  REAL, DIMENSION(100) :: u, v
  ! Instructions
  ! Récupération des donnees du fichier don.dat
  OPEN(10, file='don.dat')
  READ(10,*) n
  DO i = 1, n
    READ(10,*) u(i)
  END DO
  CLOSE(10)
  umax = u(1)
  DO i = 2, n              ! Determination du max de u
     IF( u(i) >= umax ) umax = u(i)
  END DO
  DO i = 1, n          ! Recherche du min + indice de u
     jmin = 1
     umin = umax
     DO j = 1, n
        IF ( u(j) <= umin ) THEN
           umin = u(j)
           jmin = j
        END IF
     END DO
     v(i) = umin             ! Met la valeur min dans v
     u(jmin) = umax          ! Ecrase la valeur min de u
  END DO
  OPEN(20, file='res.dat') ! Ecriture dans res.dat
  WRITE(20,*) n
  DO i = 1, n
    WRITE(20,*) v(i)
  END DO
  CLOSE(20)
END PROGRAM TRI
```

Suite de Fibonacci

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

INP ENSEEIHT

Programmation impérative:
Méthode de Programmation

3. Fonctions + Subroutines

Thomas Bonometti

Table des matières

Qu'est qu'une procédure?

Une procédure résulte du raffinement d'une action abstraite que le processeur n'est pas en mesure d'exécuter.

Une fois définie, elle devient une action élémentaire ou une "instruction" disponible pour le processeur

Une procédure manipule **des variables**:

- soit **locales** à la procédure, c'est-à-dire qu'elle seule les connait et les utilise
- soit **proviennent de l'extérieur**, c'est-à-dire qu'elles sont transmises à la procédure (passage d'arguments)

Pourquoi des procédures (sous-programmes, fonctions) ?

Compréhension globale du programme

Lecture du programme principal plus compréhensible (le nom de la procédure donne l'intention)

Compréhension détaillée du programme

Procédures « courtes » = facile à comprendre

Factorisation

Plus de copier-coller = moins d'erreurs

Mise au point facilitée

Tests faits à l'échelle d'une procédure = débuggage plus rapide

Ex.: Le diabolo menthe / fraise

Programme principal

Algorithme Niveau 1

```
Variables : Verre, sirop de fraise, sirop de
menthe, glaçon, limonade

Début Algorithme

Prendre un verre
SP Mettre du sirop dans le verre avec …
Mettre de la limonade dans le verre
Mettre un glaçon dans le verre

Fin Algorithme
```


Ex.: Le diabolo menthe / fraise

Programme principal

Algorithme Niveau 1

```
Variables : Verre, sirop de fraise, sirop de
menthe, glaçon, limonade

Début Algorithme

Prendre un verre
SP Mettre du sirop dans le verre avec …
Mettre de la limonade dans le verre
Mettre un glaçon dans le verre

Fin Algorithme
```

Sous-programme

Algorithme SP Mettre Sirop dans le verre

```
Variables : sirop de fraise(IN), sirop de
menthe(IN), verre(IN/OUT), choix (local)

Début Algorithme
Demander choix
Si choix Menthe
  Alors Mettre du sirop de Menthe dans le verre
Sinon Si choix Fraise
  Alors Mettre du sirop de fraise dans le verre
FinSi

Fin Algorithme
```

Ex.: Le diabolo menthe / fraise

Programme principal

```
Algorithme Niveau 1

Variables : Verre, sirop de fraise, sirop de
menthe, glaçon, limonade

Début Algorithme

Prendre un verre
SP Mettre du sirop dans le verre avec …
Mettre de la limonade dans le verre
Mettre un glaçon dans le verre

Fin Algorithme
```

Sous-programme

```
Algorithme SP Mettre Sirop dans le verre

Variables : sirop de fraise(IN), sirop de
menthe(IN), verre(IN/OUT), choix (local)

Début Algorithme
Demander choix
Si choix Menthe
  Alors Mettre du sirop de Menthe dans le verre
Sinon Si choix Fraise
  Alors Mettre du sirop de fraise dans le verre
FinSi

Fin Algorithme
```

Les variables dont le sous-programme a besoin pour fonctionner sont :

- Le sirop de fraise ➔ on ne veut pas modifier le sirop de fraise juste y accéder (IN)
- Le sirop de menthe ➔ on ne veut pas modifier le sirop de menthe juste y accéder (IN)
- Le verre ➔ on veut « modifier » le verre pour y mettre du sirop (IN-OUT)
- choix est une variable locale utilisées seulement par le sous-programme

© cours 3EA

FUNCTION

Un peu comme les « fonctions mathématiques »

Fournit un seul résultat

Les fonctions

Algorithmique

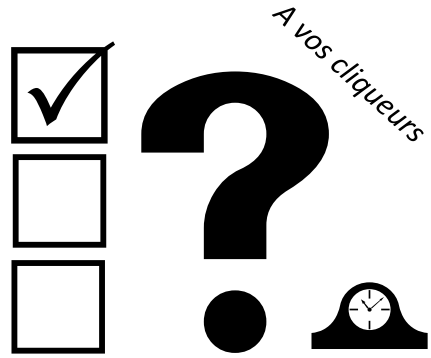
Définition de la fonction f telle que:
$f(a,b,x) = (a\ x^3 + b/x\ ) /2$
Entrée: a, b, x
Résultat: f

Nota: le symbole « f » ne pourra plus être utilisé ailleurs, il est affecté à la fonction

FORTRAN

arguments de la fonction

```
! Definition de f
FUNCTION f(a,b,x)
   IMPLICIT NONE
   REAL, INTENT (IN) :: a,b,x
   REAL :: f
   f = ( a * x**3 + b / x ) / 2.
END FUNCTION f

! Programme principal
program TOTO
implicit none

REAL :: a,b,x
REAL :: f

print*,"Donner les valeurs de a, b, x"
read*, a,b,x
print*,"f(a,b,x)=",f(a,b,x)

end program TOTO
```

Qu'affiche le programme quand l'utilisateur entre a=5 b=3 x=1 ?

Quiz

A. f(a,b,x)=f(5,3,1)

B. 4

C. f(a,b,x)=4

D. f(5,3,1)=4

E. ni A, ni B, ni C, ni D…

FORTRAN

```
! Definition de f

FUNCTION f(a,b,x)
   IMPLICIT NONE
   REAL, INTENT (IN) :: a,b,x
   REAL :: f
   f = ( a * x**3 + b / x ) / 2.
END FUNCTION f

! Programme principal
program TOTO
implicit none

REAL :: a,b,x
REAL :: f

print*,"Donner les valeurs de a, b, x"
read*, a,b,x
print*,"f(a,b,x)=",f(a,b,x)

end program TOTO
```


Exemple

- On veut écrire une fonction f fournissant en résultat la valeur de (a x^3 + b/x) /2, les valeurs de a, b et x étant fournies en argument.
- On veut écrire un programme qui affiche les valeurs de la fonction f pour x=1, 2, 3, …, 10.

FORTRAN

```
! Procédure
FUNCTION f(u,v,w)
  IMPLICIT NONE
  REAL, INTENT (IN) :: u,v,w
  REAL :: f
  f = ( u * w**3 + v / w ) / 2.
END FUNCTION f
```

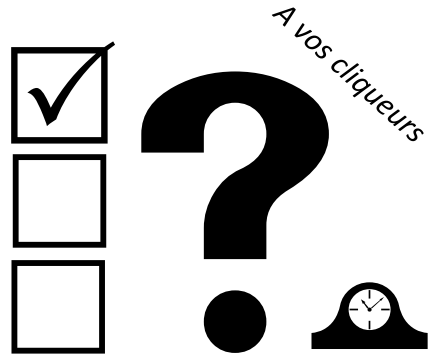
FORTRAN

```
! Programme
PROGRAM valeur_f
  IMPLICIT NONE
  REAL :: a, b, x, res
  REAL :: f
  INTEGER :: i
  a = 5.
  b = 3.
  DO i = 1, 10
    x = real(i)
    res = f(a, b, x)
    PRINT*,'Pour x= ', x, 'le resultat est ', res
  END DO
END PROGRAM valeur_f
```

Exemple

Lancement du programme - affichage à l'écran

```
Pour x =   1.0000000   le resultat est:   4.0000000
Pour x =   2.0000000   le resultat est:   20.750000
Pour x =   3.0000000   le resultat est:   68.000000
Pour x =   4.0000000   le resultat est:   160.37500
Pour x =   5.0000000   le resultat est:   312.79999
Pour x =   6.0000000   le resultat est:   540.25000
Pour x =   7.0000000   le resultat est:   857.71429
Pour x =   8.0000000   le resultat est:   1280.1875
Pour x =   9.0000000   le resultat est:   1822.6666
Pour x =   10.000000   le resultat est:   2500.1499
```

Un autre exemple

Quiz

- Quel type de résultat donne la fonction ?
- Que fait la fonction ?
- Que fait le programme pour age=15 ?
- Que fait le programme pour age=19 ?

```
! Function
FUNCTION test_majorite(toto)

   IMPLICIT NONE
   REAL, INTENT (IN) :: toto
   LOGICAL :: test_majorite

   IF ( toto >= 18. ) THEN
       test_majorite = .TRUE.
   ELSE
        test_majorite = .FALSE.
   END IF

END FUNCTION test_majorite

! Programme
PROGRAM entree_disco

   IMPLICIT NONE
   REAL :: age
   LOGICAL :: test_majorite

   PRINT*, 'Donner son age'
   READ*, age
   IF (test_majorite(age) .eqv. .TRUE.) THEN
       PRINT*, 'Bonne soiree'
   ELSE
        PRINT*, 'Revenez dans' , 18 - age , 'ans'
   END IF

END PROGRAM entree_disco
```


Fonctions intrinsèques de Fortran 90

FORTRAN (liste non-hexaustive)

! Fonctions mathématiques usuelles

cos(a), acos(a), sin(a), asin(a), tan(a), atan(a), exp(a), log(a), log10(a), sqrt(a)

! Fonctions numériques

abs(a)	! Fournit la valeur absolue de a (entier, réel)
ceiling(a)	! Fournit l'entier immédiatement supérieur à la valeur du réel a
floor(a)	! Fournit l'entier immédiatement inférieur à la valeur du réel a
int(a)	! Convertit a en entier
real(a)	! Convertit a en réel
max(a1, a2, …, an)	! Fournit la valeur max des ai (entiers ou réels)
min(a1, a2, …, an)	! Fournit la valeur min des ai (entiers ou réels)
mod(a,p)	! Fournit a-int(a/p)*p, le reste de la division euclidienne de a/p
random_number(a)	! Fournit dans a un nombre pseudo-aléatoire tel que $0 \leq a < 1$

Fonctions intrinsèques de Fortran 90

(suite)

```
! Fonctions relatives aux tableaux
dot_product(u,v)     ! Fournit le produit scalaire du vecteur u par le vecteur v
maxval(tab)          ! Fournit la valeur max des éléments du tableau tab
minval(tab)          ! Fournit la valeur min des éléments du tableau tab
sum(tab)             ! Fournit la valeur de la somme des éléments du tableau tab
maxloc(tab)          ! Fournit les indices de l'élément de valeur max du tableau tab
minloc(tab)          ! Fournit les indices de l'élément de valeur min du tableau tab
```


Exercice

- Ecrire une fonction fournissant en résultat le volume d'une sphère dont le rayon (de type real) lui est fourni en argument.

- Ecrire le programme l'utilisant pour calculer 3 volumes correspondant à trois rayons fournis en donnée (au clavier)

SUBROUTINE

= sous-programme avec plusieurs Entrées – Sorties

Ne fournit pas qu'un seul résultat

SUBROUTINE

FORTRAN

```
! Exemple de programme de facture
PROGRAM facture

  IMPLICIT NONE
  INTEGER :: n
  REAL :: pht, ptht, pttc

  WRITE(*,*) 'Donner n et le prix unitaire HT'
  READ(*,*) n, pht

  CALL prix(n, pht, ptht, pttc)

  WRITE(*,*) 'Le prix total HT est', ptht
  WRITE(*,*) 'Le prix total TTC est', pttc

END PROGRAM facture
```

Variables globales

FORTRAN

```
SUBROUTINE prix( n,  pht,  ptht,  pttc )

  IMPLICIT NONE

  INTEGER, INTENT(IN) :: n
  REAL, INTENT(IN) :: pht
  REAL, INTENT(OUT) :: ptht, pttc
  REAL :: taxe

  taxe = 1.186
  ptht = real(n) * pht
  pttc = ptht * taxe

END SUBROUTINE prix
```

données

résultats

Variables en arguments

variable locale

Genre des arguments

INTENT(IN) = la valeur peut être utilisée mais pas modifiée dans la procédure

INTENT(OUT) = la valeur peut être modifiée mais pas utilisée dans la procédure

INTENT(INOUT) = la valeur peut être utilisée et modifiée dans la procédure

En général, les variables de données sont du genre INTENT(IN)
les variables résultats sont du genre INTENT(OUT)

Exception: les variables modifiées **avant** la procédure et **dans** la procédure sont du genre INTENT(INOUT)

Exemple du tri (4)

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 fonction qui détermine la valeur maximum du vecteur

- plusieurs sous-programmes qui, respectivement:
  - -> lit les valeurs du vecteur non-trié dans don.dat
  - -> détermine la valeur et l'emplacement du minimum du vecteur
  - -> écrit les valeurs du vecteur trié dans res.dat

Exemple du tri (4)

```
PROGRAM TRI
   IMPLICIT NONE
   ! DECLARATIONS
   INTEGER :: n
   REAL, DIMENSION(100) :: u, v

   ! INSTRUCTIONS
   CALL lecture_valeur('don.dat',n)
   CALL lecture_vecteur('don.dat',n,u)
   CALL tri_vecteur(n,u,v)
   CALL ecriture('res.dat',n,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(100), 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 (4)

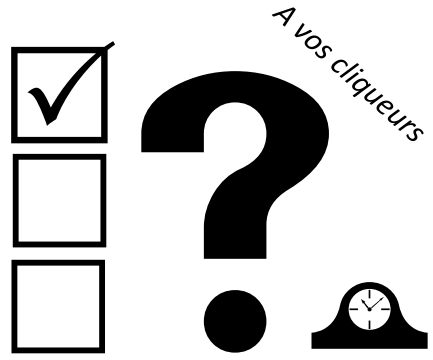
```
!----------------------------
SUBROUTINE tri_vecteur(n,u,v)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(100), INTENT(INOUT)  :: u
   REAL, DIMENSION(100), 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(100), 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 (4)

```
!-------------------------------------------
SUBROUTINE recherche_min(n,u,umax,umin,jmin)

   IMPLICIT NONE
   ! DECLARATIONS
   ! variables en arguments
   INTEGER, INTENT(IN) :: n
   REAL, DIMENSION(100), 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(100), 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
```



Que mettre en arguments ?

Quiz n°1

Quelle version est la bonne?

! Version n°1

```
SUBROUTINE toto (a, f)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: a
  INTEGER, INTENT(OUT) :: f
  INTEGER :: i
  DO i = 1, a
     f = b + c
  END DO
END SUBROUTINE toto
```

! Version n°2

```
SUBROUTINE toto (a, b, c, f)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: a, b, c
  INTEGER, INTENT(OUT) :  f
  INTEGER :: i
  DO i = 1, a
     f = b + c
  END DO
END SUBROUTINE toto
```

! Version n°3

```
SUBROUTINE toto (a, b, c, d, e, f)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: a, b, c, d
  INTEGER, INTENT(OUT) :: f
  INTEGER :: i
  DO i = 1, a
     f = b + c
  END DO
END SUBROUTINE toto
```

Qu'affiche ce programme ?

Quiz n°2

```
PROGRAM  tete_a_toto

   IMPLICIT NONE

   INTEGER :: f

   CALL toto ( 1, 1, 1, f)
   WRITE(*,*) 'f=',f

END PROGRAM  tete_a_toto


SUBROUTINE toto (a, b, c, f)

   IMPLICIT NONE

   INTEGER, INTENT(IN) :: a, b, c
   INTEGER, INTENT(OUT) ::  f
   INTEGER :: i

   DO i = 1, a
        f = b + c
   END DO

END SUBROUTINE toto
```

A. Rien…

B. f=1

C. f=2

D. f= (une valeur aléatoire)

E. ni A, ni B, ni C, ni D…

Recommandations (subroutines)

- Eviter de mélanger dans le sous-programme (SP) traitement (calcul) et interactions avec l'utilisateur du programme (affichage ou saisie) : Les traitements sont plus stables que l'IHM…
- Un SP doit être une boîte noire
  ⇒ ne pas dépendre de variables globales
- Un SP ne doit pas avoir trop de paramètres
  ⇒ regrouper les paramètres dans une structure
- Un SP ne doit pas être trop long (sinon le découper)
- Un SP ne doit pas avoir trop de structures de contrôle imbriquées (sinon faire des SP dans le SP).

Ce qu'on peut faire

- Mettre plusieurs subroutines dans une subroutine
- Mettre plusieurs fonctions dans une fonction
- Mettre plusieurs fonctions dans un subroutine

- Appeler plusieurs fois une subroutine (n'importe où)
- Appeler plusieurs fois une fonction (n'importe où)

Exercices

Suite de Fibonacci

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

On utilise pour cela plusieurs sous-programmes qui, respectivement:

-> lit don_entree.dat
-> calcule les u(i)
-> écrit les résultats dans don_sortie.dat

INP ENSEEIHT

Programmation impérative:
Méthode de Programmation

4. Compilation avec Makefile + Allocation dynamique

Thomas Bonometti

Table des matières

INP 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

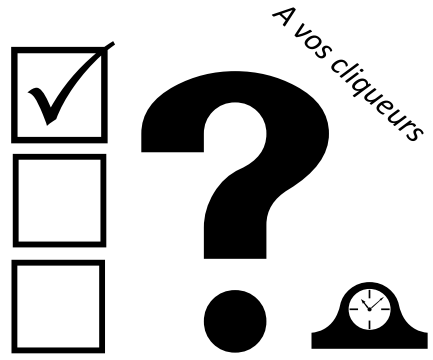
1 module.o: …

2 programme_principal.o: …

3 subroutines.o: …

Plan du cours

Allocation dynamique



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

#QDLE#Q#ABCD*E#60#

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

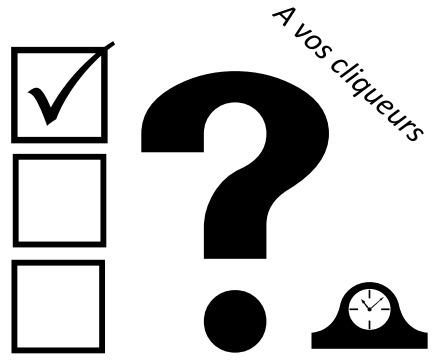
- en n'allouant que l'espace nécessaire

- 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)



Quiz

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

INP ENSEEIHT TOULOUSE

Programmation impérative: Méthode de Programmation

5. Structures de données

Thomas Bonometti

Les structures, pour quoi faire ?

INP ENSEEIHT TOULOUSE

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

Création de structures

Algorithmique	FORTRAN
Création d'une structure client nom prenom valeur	! définition TYPE client character (len=20) :: nom character (len=10) :: prenom real :: valeur END TYPE client

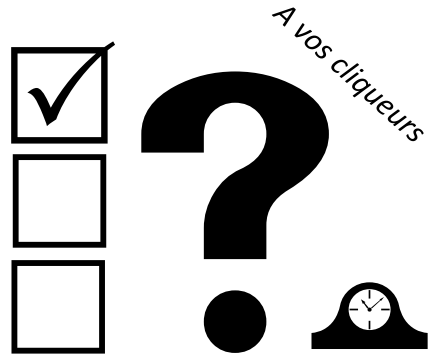
Une structure est comme un « panier » contenant différents objets (ou champs) ayant un lien logique entre eux

Utilisation de structures

Algorithmique	FORTRAN
Déclaration d'une structure	! déclaration de variables TYPE (client) :: cl1
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 »	! 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

#QDLE#Q#ABC*D#30#

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
```

Structures contenant un tableau

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
```


Structures contenant une matrice

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
```

Exercice

Algorithmique

FORTRAN

Exercice:

définir les différentes structures
(avec plusieurs variantes…)

date
- jour
- mois
- an

client
- nom
- prenom
- date_de_naissance
- compte

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 ≠ 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

Exemple du tri (6)

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
- 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
```


Exercices

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 $u_1, u_2, \ldots, u_n$ ($n \leq 100$ 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 u_1 u_2 » 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) »

INP ENSEEIHT

Programmation impérative: Méthode de Programmation

6. Modules + Formats

Thomas Bonometti

Table des matières

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**

nb d'entiers — INTEGER — nb total de caractères

Espace: **X**

Réels: **n F w . d**

nb de réels — FLOAT

nb total de caractères — nb de chiffres après le point décimal

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

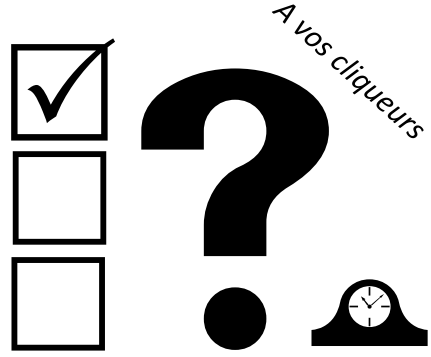
```
tbonomet@perche:~/FORTRAN90$ ./prog.exe
        2013
  2013
 2013
2013
***
        2013      123456
    2013  123456
   2013 123456
  2013123456
 2013*****
  2013 123456
  2013 123456
  2013 123456
tbonomet@perche:~/FORTRAN90$
```

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$
```

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