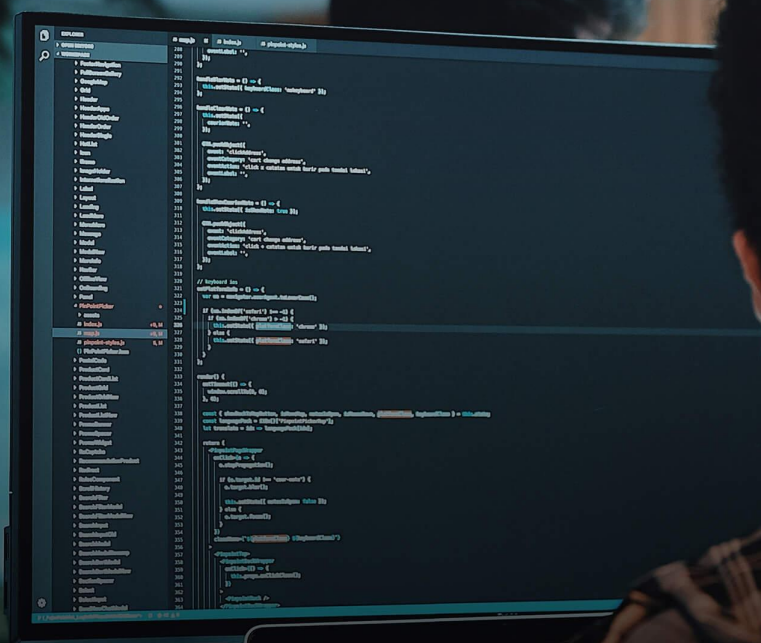


# ALGORITHMIQUE

Pr. Hamid MRAOUI

# 2



SMI S3 2018/2019

fso.umpoujda.com

---

# Algorithmique 2

---

## DESCRIPTIF DU MODULE ALGORITMIQUE II

---

- Chapitre 1 : Fonctions et procédures.
- Chapitre 2 : La récursivité.
- Chapitre 3 : Enregistrements et fichiers.
- Chapitre 4 : La complexité.
- Chapitre 5 : Preuves d'algorithmes.

### VOLUME HORAIRE

✓ COURS : 1H 30 MIN

✓ TD : 1H 30 MIN

---

## Rappel : Algorithmique 1

---

### Qu'est ce qu' un algorithme ?

Un algorithme est une suite finie d'**instructions à appliquer** dans un ordre déterminé à un nombre fini de **données** pour arriver, en un nombre fini d'étapes, à un certain **résultat**, et cela indépendamment des données

### Pourquoi utiliser un algorithme ?

---

Un algorithme décrit ce qui doit faire l'**ordinateur** pour arriver un but bien précis. Ce sont les **instructions** qu'on doit lui donner. Donc l'algorithme est un moyen pour le **programmeur** de présenter son **approche** d'un problème donné à d'autres personnes, dans un **langage clair** et **compréhensible** par l'être humain.

### Définition d'un algorithme

---

C'est un pseudo-langage qui est conçu pour résoudre les problèmes et applications sans aucune contrainte due aux langages de programmation et aux spécificités de la machine. Ce pseudo-langage sera ensuite traduit et codé dans le langage de programmation désiré.

## Structure générale d'un algorithme

### + Titre du Problème

#### + Déclaration des Objets

- ✓ Déclaration des Constantes
- ✓ Déclaration des Variables
- ✓ Déclaration des Tableaux
- ✓ Déclaration des Procédures et Fonctions

#### + Manipulation

Début

*Actions*

FIN

## Conception d'un algorithme

### • Les étapes de conception d'un algorithme :

+ Comprendre le problème;

+ Identifier les données du départ (entrées) et celle(s) qu'il faut obtenir (sorties);

+ Structurer les données (variables ou constantes, type...);

+ Déterminer les transformations nécessaires à faire pour obtenir les résultats (traitements/développements) ;

+ Présenter les résultats.

## Caractéristiques d'un algorithme

Un bon algorithme doit être

- + **Lisibles** (Compréhensible).
- + **De haut niveau** (être traduit en un langage).
- + **Précis** (Pas de confusion).
- + **Concis** (ne doit pas dépasser une page).
- + **Structuré** (Parties facilement identifiables).

## L'algorithme sous forme de texte

**Algorithme** Nom\_de\_algorithme

**Variable** /\* Déclaration des variables \*/

- **DEBUT**

- Instruction 1
- Instruction 2
- .....
- .....
- Instruction n

- **FIN**

## Déclaration des variables

Pour exister une variable doit être déclarée, c'est –à- dire que vous devez indiquer au début de l'algorithme comment elle s'appelle et ce qu'elle doit contenir. Les variables se déclarent au début de l'algorithme, avant le programme lui-même mais après le mot

« **Variable** »

**Variable**

Variable1 : type

variable2,variable3,... : type

## Les types

- Les constantes : désignent des références à des valeurs invariantes dans le programme
- Les entiers : nombres sans virgule, négatifs ou positifs.
- Les réels : nombres à virgule, positifs ou négatifs.
- Les booléens : Pour déterminer si une affirmation vraie ou fausse
- Les caractères : pour représenter un seul caractère.
- Les chaînes : ce sont une suite de caractères.
- Les tableaux : permettent de représenter un ensemble de valeurs appartenant toutes au même type.

## Les types

### Syntaxe de la déclaration :

**Constante**    Nom\_Constante ← Valeur

**Variable**    variable1,variable2,... : Entier

**Variable**    variable1,variable2,... : Réel

**Variable**    variable1,variable2,... : Caractère

**Variable**    variable1,variable2,... : Chaîne

**Variable**    Tab1,... : tableau[0..nbelement-1] d'entiers

**Variable**    Tab1,... : tableau[0..dim1-1] [0..dim2-1] de réels

## Saisie et affichage

Pour afficher un message à l'écran, il faut utiliser la pseudo-instruction « **Afficher** ».

```
Algorithme Afficher
Début
    Afficher ("Bonjour")
Fin
```



## Saisie et affichage

Pour inviter un utilisateur à rentrer au clavier une valeur, utiliser le mot « **Lire** ».

```
Algorithme Lire
Variable x : Entier
Début
    Afficher("Saisir un entier x ")
    Lire (x)
    Afficher (" x vaut :", x )
Fin
```

## Opérateurs et calculs

← Symbole d'affectation

Pour affecter une valeur à une variable, on écrit :

Variable ← Valeur

### Opérateurs et calculs

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
% ou mod	Modulo : le reste de la division de 2 valeurs entières

### Opérateurs et calculs

Opérateur	Signification
=	Égal
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal
<>	différent

## Opérateurs et calculs

Opérateur	Signification
et	Et logique
ou	Ou logique
non	Négation logique
	.....

## Tests et conditions

- L'instruction Si :

**Si** Booléen **alors**

Bloc d'instructions

[ **Sinon**

Bloc d'instructions } ← *Option Facultative*

**Finsi**

## Tests et conditions

- **Algorithme** Structure\_Alrenative\_1
- **Variable** x : entier
- **Début**
- **Afficher** ("Saisir un entier x ")
- **Lire** (x)
- **Si** (x > 0) **alors**
- **Afficher**("x est un nombre positif ")
- **Finsi**
- **Fin**

## Tests et conditions

- **Algorithme** Structure\_Alrenative\_2
- **Variable** x : Entier
- **Début**
- **Afficher** ("Saisir un entier x ")
- **Lire** (x)
- **Si** (x > 0) **alors**
- **Afficher** (" x est un nombre positif ")
- **Sinon**
- **Afficher** (" x est un nombre négatif ou nul")
- **Finsi**
- **Fin**

## Tests et conditions

- **Algorithme** Maximum
- **Variable**  
a ,b, max : Entier
- **Début**
- **Afficher** ("Saisir deux entiers a et b ")
- **Lire**(a, b)
- **Si** (a > b) **alors**
- **max** ← a
- **Sinon**
- **max** ← b
- **Finsi**
- **Afficher** ("le maximum de " , a , " et de " , b, " est : " , max)
- **Fin**

H. Miraoui

23

## Choix multiples

- **Suivant Cas** variable **Faire**
- **Cas** Valeur 1 : *Actions 1; sortir*
- **Cas** Valeur 2 : *Action 2; sortir*
- ..
- ..
- **Autre** : *Action par défaut*
- **Fin Suivant**

H. Miraoui

24

## Les boucles

- L'instruction Pour :

**<Initialisation>**

**Pour** variable allant de valeur1 à valeur2 **faire**

Bloc d'instructions

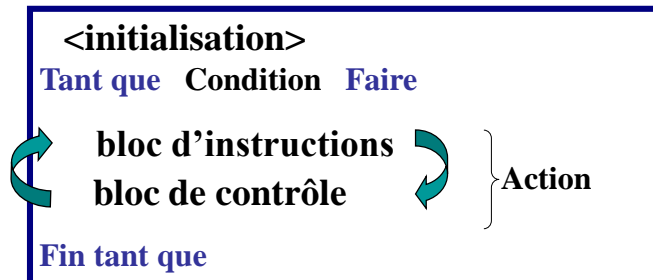
**Fin Pour**

## Les boucles

- **Algorithme Factoriel**
- **Variable**
  - **N : Entier**
  - **i : Entier**
  - **Fact : Entier**
- **Début**
  - **Afficher** (" Saisir une valeur entière N > 0: ")
  - **Lire** (N )
  - **Fact** ← 1
  - **Pour** i allant de 1 à N **Faire**
    - **Fact**← Fact\*i
  - **Fin Pour**
  - **Afficher** (" Le factoriel de ", N , " est : " , Fact)
- **Fin**

## Les boucles

- L'instruction Tant que :



## Les boucles

- Variable
- $i : \text{Entier}$
- Début
- $i \leftarrow 0$  // Initialisation
- Tant que  $(i < 300)$  Faire
- Afficher(" Bonjour tout le monde ")
- $i \leftarrow i + 1$
- Fin tant que
- Fin

## Les boucles

### L'instruction Faire jusqu'à :

<Initialisation>

**Faire**

bloc d'instructions

**Jusqu'à**      **Condition**

## Les boucles

**Titre :** Boucle3

**Variable**

i , y : Entier

**Début**

i ← 2

y ← 0

**Faire**

i ← i+1

y ← y+i

**Afficher** (" y = " , y)

**Jusqu'à** (i = 7)

**Fin**

*Donner les valeurs de y*



## Les boucles

### Variable

N,S, i : Entier

### Début

Afficher ("Saisir une valeur entière positive :")

Lire (N)

S ← 0

i ← 0

### Faire

i ← i + 1

S ← S + i

jusqu'à (i ≥ N)

Afficher ("La somme : S = ", S)

FIN

## Les fonctions

### Variable

a,b, c : Réel

max2,max3 : Réel

### Début

Afficher ("Saisir 3 nombre réels : ")

Lire (a,b,c)

Si (a > b) alors

max2 ← a

Sinon

max2 ← b

Finsi

Si (c > max2) alors

max3 ← c

Sinon

max3 ← max2

Finsi

Afficher ("Le maximum est = ", max3)

FIN

## Les fonctions

```
Variable
  a,b, c : Réel
  max2,max3 : Réel
Début
  Afficher ("Saisir 3 nombre réels : ")
  Lire (a,b,c)
  Si (a > b) alors
    max2 ← a
  Sinon
    max2 ← b
  Finsi
  Si (c > max2) alors
    max3 ← c
  Sinon
    max3 ← max2
  Finsi
  Afficher ("Le maximum est = ", max3)
FIN
```

H. Mraoui

33

## Les fonctions

- **Problème** : Dès qu'on commence à écrire des programmes, il devient difficile d'avoir une vision globale sur son fonctionnement.

✚ Difficulté de trouver des erreurs.

✚ Redondance

✚ ....

- **Solution** : Décomposer le problème en sous problèmes

H. Mraoui

34

## Les fonctions

**Fonction** Maximum2(x : Réel, y : Réel) : Réel

**Variable**

max2 : Réel

**DébutFonction**

**Si** (x > y) **alors**

max2 ← x

**Sinon**

max2 ← y

**Finsi**

**Retourne** max2

**FinFonction**

## Les fonctions

**Algorithme** Maximum\_de\_trois\_variable\_1

**Variable**

a,b, c : Réel

max2,max3 : Réel

**Début**

**Afficher** ("Saisir 3 nombre réels : ")

**Lire** (a,b,c)

max2 ← Maximum2(a,b)

max3 ← Maximum2(max2,c)

**Afficher**("Le maximum est = ", max3)

**FIN**

## Les fonctions

**Fonction** Maximum3(x : Réel, y : Réel, z : Réel) : Réel

**Variable**

max3 : Réel

**DébutFonction**

max3 ← Maximum2(Maximum2(x ,y) ,z)

Retourne max3

**FinFonction**

## Les fonctions

**Algorithme** Maximum\_de\_trois\_variable\_2

**Variable**

a,b, c : Réel

max3 : Réel

**Début**

Afficher ("Saisir 3 nombre réels : ")

Lire (a,b,c)

max3 ← Maximum3(a,b,c)

Afficher ("Le maximum est = ", max3)

**FIN**

## Les fonctions

---

**Une fonction est un sous programme particulier, accomplissant une tâche particulière et qui ne renvoie, dans l'algorithme principal, qu'un et un seul résultat.**

**Pourquoi on l'utilise :**

- + Décomposer l'algorithme en de parties appelées par le programme principal.
- + Eviter la répétition inutile les mêmes instructions plusieurs fois.

## Les fonctions

---

**Une fonction est un sous-programme qui :**

- + A un nom.
- + Peut avoir des paramètres ou arguments.
- + Retourne une valeur d'un certain type.
- + Peut avoir besoin de variables.
- + Est composé d'instructions.

**Remarque : une fonction peut ne pas avoir de paramètres.**

## Les fonctions

**La fonction peut être déclarée de la manière suivante :**

**Fonction** Nom\_de\_fonction(Liste des arguments) :  
type\_de\_retour

**Variable** /\* Déclaration des variables \*/

- **DébutFonction**
- **Liste des instructions de la fonction**
- **Retourne (Résultat)**
- **FinFonction**

## Les fonctions

**Exemple : Fonction qui retourne le carré d'un entier :**

**Fonction** carré\_d'un\_entier(a : Entier) : Entier

**Variable**

**Résultat** : Entier

**DébutFonction**

**Résultat** ← a\*a


**Retourne (Résultat)**

**FinFonction**

### Variables locales

- L'endroit où les variables sont déclarées est très important. Selon cet endroit les variables ont une **portée différente**. La portée d'une variable est sa visibilité au sein des différentes parties du programme.
- Les variables accessibles uniquement par le programme ou sous-programme dans lesquels elles sont déclarées, sont appelées des **variables locales**.

### Variables globales

- Il serait pourtant très pratique de pouvoir accéder à une variable depuis n'importe quel endroit du programme, qu'il soit principal ou un sous-programme. La portée d'une telle variable s'étendrait à tout le code. Ce type de variable s'appelle une **variable globale**.
- Les variables globales sont déclarées de cette manière (en dehors des sous-programmes et du programme) :  **Variable Globales**  
**a : Entier**

## Appel d'une fonction

- **L'appel** : c'est l'utilisation d'une fonction à l'intérieur d'une autre fonction ou de l'algorithme principal.

Voici deux exemples :

**var** ← **Nom\_de\_fonction**(var1, var2, ....)

**Afficher** (**Nom\_de\_fonction**(var1, var2, ....))

## Arguments d'une fonction

- Les arguments servent à échanger des données entre l'algorithme principal et les fonctions.
- Les arguments placés dans la déclaration d'une fonction sont des **variables locales** du sous-programme.
- Les arguments, placés dans l'appel d'une fonction, contiennent les valeurs pour effectuer le traitement.
- Le nombre d'arguments dans l'appel d'une fonction doit être égal au nombre d'arguments d'entrée. L'ordre et le type des arguments doivent correspondre.



### Fonctions : cas des tableaux

- On propose d'écrire une fonction qui renvoie l'indice de la valeur maximale d'un tableau.
- Écrire une fonction qui prend en argument un tableau et permet le trier par ordre décroissant.
- Écrire l'algorithme principal.

### Fonctions : cas des tableaux

**Fonction** `indice_val_max` (T : tableau[ ] d'entiers, taille : Entier, k : Entier) : Entier

**Variable**

i, imax, max2 : Entier

• **DébutFonction**

max2 ← T[k]

imax ← k

• **Pour** i allant de k+1 à taille-1 **Faire**

• **DébutPour**

• **Si** ( T[i]>max2) **Alors**

max2 ← T[i]

imax ← i

**FinSi**

**FinPour**

**Retourne** imax

• **FinFonction**

## Fonctions : cas des tableaux

**Fonction** **Tri\_décroissant** (T : tableau[ ] d'entiers, taille : Entier) : tableau d'entiers

**Variable**

i, imax, a, k : Entier

• **DébutFonction**

• k ← 0

• **Pour** i allant de taille à 2 (Pas -1) **Faire**

• **DébutPour**

• imax ← indice\_val\_max(T, taille, k)

• a ← T[k]

• T[k] ← T[imax]

• T[imax] ← a

• k ← k+1

• **FinPour**

• **Retourne** T

• **FinFonction**

H. Mraoui

49

## Fonctions : cas des tableaux

**Algorithme** Programme\_Pricipal

**Constante** n ← 6

**Variable**

i: Entier

T : tableau[ 0..n-1] d'entiers

**Début**

• **Pour** i allant de 0 à n-1 **Faire**

• **DébutPour**

• Lire (T[i])

• **FinPour**

• T ← Tri\_décroissant (T,n)

• **Pour** i allant de 0 à n-1 **Faire**

• **DébutPour**

• Afficher (T[i])

• **FinPour**

**Fin**

H. Mraoui

50

## Procédures

- Dans certains cas, on peut avoir besoin de répéter une tâche dans plusieurs endroits, mais que dans cette tâche on ne calcule pas de résultats ou qu'on calcule plusieurs résultats à la fois. Dans ce cas on ne peut pas utiliser une fonction, on utilise une **procédure**.
- Une procédure est un sous programme semblable à une fonction mais qui retourne rien.

## Procédures

La procédure s'écrit en dehors de l'algorithme principal sous la forme :

**Procédure** Nom\_de\_procédure(Liste des arguments)

**Variable** /\* Déclaration des variables \*/

- **DébutProcédure**
- Liste des instructions de la procédure
- **FinProcédure**

## La récursivité

Un sous-programme récursif est un sous-programme qui peut s'appeler lui-même.

Il existe deux types de récursivité :

✚ **Directe ou simple** : le sous-programme s'appelle lui-même.

✚ **Indirecte ou croisée** : deux sous-programmes s'appellent l'un l'autre : le premier appelle le second, qui appelle le premier, etc.

## La récursivité

La récursivité peut être appliquée tant aux fonctions qu'aux procédures.

Modèle de fonction récursive directe :

**Fonction** recursive ( )

**Variable** /\* Déclaration des variables \*/

• **DébutFonction**

/\* instructions \*/

recursive ( )

/\* instructions \*/

• **FinFonction**

## La récursivité

Modèle de fonction récursive indirecte :

**Fonction** recursiveA ( )

**DébutFonction**

**/\*instructions \*/**

**recursiveB( )**

**/\*instructions \*/**

**FinFonction**

**Fonction** recursiveB ( )

**DébutFonction**

**/\*instructions \*/**

**recursiveA( )**

**/\*instructions \*/**

**FinFonction**

## La récursivité

Une fonction récursive contient un ou plusieurs paramètres qui évoluent lorsqu' on appelle la fonction jusqu'à satisfaire un test qui nous permettra de sortir de la fonction. La récursivité solutionne un problème en résolvant le même problème mais donne une solution plus simple. Le processus de simplification se poursuit jusqu'à l'atteinte d'un cas où la solution est connue.

## La récursivité

```
Fonction recursive ( )  
Variable /* Déclaration des variables */  
• DébutFonction  
    /* instructions */  
    Si (test d'arrêt) Alors  
        /* instructions */  
    Sinon  
        recursive ( )  
    Finsi  
    /* instructions */  
• FinFonction
```

## La récursivité

Une factorielle est l'exemple classique d'application d'un algorithme récursif.

### Calcul de la factorielle :

$$\text{Fact}(n) = \begin{cases} 1 & \text{Si } n=0, \\ \text{Fact}(n-1)*n & \text{Si } n>0 \end{cases}$$

## La récursivité

**Fonction** Fact ( n : Entier ) : Entier

- **DébutFonction**

**Si** ( n = 0 ) **Alors**

**Retourne** 1

**Sinon**

**Retourne** n\*Fact( n-1 )

**Finsi**

- **FinFonction**

## La récursivité

**Algorithme** Factorielle\_d'un\_nombre

**Variable**

    n : Entier

**Début**

**Afficher** ("Saisir un nombre entier : ")

**Lire** (n)

**Afficher** ("La factorielle de ",n, " est " , Fact(n))

**FIN**

## La récursivité

### Calcul de la suite de Fibonacci :

$$\text{Fib}(n) = \begin{cases} 1 & \text{Si } n=0, \\ 1 & \text{Si } n=1, \\ \text{Fib}(n-1)+\text{Fib}(n-2) & \text{Si } n>1 \end{cases}$$

## La récursivité

**Fonction Fib ( n : Entier ) : Entier**

- **DébutFonction**

**Si ( n = 0 ) Alors**

        Retourne 1

**Sinon**

**Si ( n = 1 ) Alors**

            Retourne 1

**Sinon**

            Retourne Fib( n-1 )+Fib(n-2)

**Finsi**

**Finsi**

- **FinFonction**



## La récursivité

### Algorithme Appel\_Fonction\_Fibonacci

#### Variable

**n** : Entier

#### Début

**Afficher** ("Saisir un nombre entier : ")

**Lire** (n)

**Afficher** ("La factorielle de ", n, " est " , **Fib**(n))

**FIN**

## La récursivité : Exemple

- Produit des éléments d'un tableau **T**.
- La taille du problème est liée à la taille du tableau : **n**.
- Multiplier les éléments d'un tableau de taille **n** revient à multiplier **T[ind\_initial]** avec le produit du reste (tableau de taille **n-1**).
- Le produit s'arrête si on vérifie que **ind\_initial > ind\_final**

## La récursivité : Exemple

**Fonction Produit**(T : tableau[ ] d'entiers, ind\_min: Entier, ind\_max : Entier) : Entier

- **DébutFonction**

- Si (ind\_min=ind\_max) alors /\* Point d'arrêt \*/
- retourne T[ind\_max]
- Sinon /\* Appel récursif \*/
- retourne T[ind\_min]\***Produit** (T,ind\_min+1,ind\_max)
- Finsi
- **FinFonction**

## La récursivité

### Recherche dichotomique :

On suppose qu'on dispose d'un tableau et on se donne un élément quelconque et on cherche si cet élément est dans le tableau ou non.

**Attention :** uniquement si le tableau est déjà trié.

## La récursivité

A chaque étape :

- Tester si le tableau est vide (Arrêt des appels récursifs avec échec)
- Calculer l'indice moyen  $(\text{indice\_max} + \text{indice\_min}) / 2$
- Comparer la valeur présente à l'indice moyen avec l'élément recherché :
  - 1) Si l'élément recherché est à l'indice moyen (arrêt succès).
  - 2) Si l'élément est supérieur à la valeur `tableau[indice_moyen]` relancer la recherche avec le tableau supérieur.
  - 3) Sinon relancer la recherche avec le tableau inférieur.

H. Mraoui

67

## La récursivité

**Fonction Recherche\_Dicho**(T : tableau[ ] d'entiers, taille : Entier, x: Entier) : booléen

**Variable**

ind\_min, ind\_max, ind\_moyen : Entier

trouvé : Booléen

• **DébutFonction**

```
• ind_min ← 0
• ind_max ← taille-1
• trouvé ← Faux
• Tant que (ind_min < ind_max) et (trouvé = Faux) Faire
•   ind_moyen ← (ind_min + ind_max) div 2
•   Si (T[ind_moyen] = x) Alors
•     trouvé ← Vrai
•   Sinon
•     Si (T[ind_moyen] > x) Alors
•       ind_max = ind_moyen - 1
•     Sinon
•       ind_min = ind_moyen + 1
```

H. Mraoui

68

## La récursivité

**Finsi**

**Finsi**

**FinTantque**

**Retourne trouvé**

- **FinFonction**

## La récursivité

**Algorithme Appel\_Fonction\_RechercheDicho**

**Constante**     $n \leftarrow 6$

**Variable**

**i,x:** Entier

**T :** tableau[ 0..n-1] d'entiers

**Début**

**Afficher** (" Donner un tableau ordonné de taille " , n)

- **Pour** i allant de 0 à n-1 **Faire**
- **DébutPour**
- **Lire** (T[i])
- **FinPour**
- **Afficher** (" Donner un entier : " )
- **Lire** (x)
- **Afficher** (" Résultat de la recherche est " , **Recherche\_Dicho**(T,n,x))

**Fin**

## La récursivité

**Fonction Recherche\_Dicho**(T : tableau[ ] d'entiers, ind\_min: Entier, ind\_max : Entier, x: Entier) : booléen

**Variable**

ind\_moyen : Entier

• **DébutFonction**

```

•           Si (ind_min<=ind_max) alors
•
•               ind_moyen ← (ind_min+ind_max) div 2
•
•               Si ( T[ind_moyen]=x) Alors
•
•                   retourne Vrai
•
•               Sinon
•
•                   Si ( T[ind_moyen]>x) Alors
•
•                       retourne Recherche_Dicho(T,ind_min,ind_moy-1,x)
•
•                   Sinon
•
•                       retourne Recherche_Dicho(T,ind_moy+1,ind_max,x)
•
•               Finsi
•
•           Finsi
•
•       Sinon
•
•           retourne Faux

```

H. Mraoui

71

## La récursivité

**Finsi**

• **FinFonction**

**Algorithme Appel\_Fonction\_RechercheDicho**

**Constante** n ← 6

**Variable**

i,x: Entier

T : tableau[ 0..n-1] d'entiers

**Début**

**Afficher** (" Donner un tableau ordonné de taille " , n)

```

•   Pour i allant de 0 à n-1 Faire
•
•       DébutPour
•
•           Lire (T[i])
•
•       FinPour
•
•   Afficher (" Donner un entier : " )
•
•       Lire (x)
•
•   Afficher (" Résultat de la recherche est " , Recherche_Dicho(T,0,n-1,x))

```

**Fin**

H. Mraoui

72

## La récursivité : Exercices

### • Exercice 1 :

Nous voulons un sous-algorithme récursif **Difference(A,k,B,C)** qui nous renvoie la différence des deux tableaux qui lui sont passés en argument (La différence de **A** et de **B**, est le tableau des éléments de **A** n'appartenant pas à **B**). Distinguer :

1. Cas des tableaux non triés.
2. Cas des tableaux triés.

## La récursivité : Exercices

**Procédure** **Différence**(A,B,C : tableau | d'entiers, k,n,m: Entier)

*/\* n la taille de A et m la taille de B\*. Cas des tableaux non triés \*/*

**DébutProcédure**

**Si** k < n **Alors**

**Si** Recherche(B, m, A[k])=Faux **Alors**

**Taille**(C) ← **Taille**(C)+1

C[**Taille**(C)-1] ← A[k]

**Finsi**

**Différence**(A,B,C,k+1,n,m)

**Finsi**

**FinProcédure**

## La récursivité : Exercices

**Fonction** **Différence**(A,B,C : tableau[ ] d'entiers, a,b,n,m: Entier) :  
tableau d'entiers

*/\* n la taille de A et m la taille de B\*. Cas des tableaux triés /*

**DébutFonction**

**Si**  $a \geq n$  **Alors**

**retourne** C

**Finsi**

**Si**  $A[a] = B[b]$  **Alors**

**retourne** **Différence**(A,B,C,a+1,b+1,n,m)

**Sinon**

**Si**  $A[a] < B[b]$  **Alors**

**Taille**(C)  $\leftarrow$  **Taille**(C)+1

C[**Taille**(C)-1]  $\leftarrow$  A[a]

H. Mraoui

75

## La récursivité : Exercices

**Retourne** **Différence**(A,B,C,a+1,b,n,m)

**Sinon**

**Retourne** **Différence**(A,B,C,a,b+1,n,m)

**Finsi**

**Finsi**

**FinFonction**

### Remarque :

L'appel initial est **Différence**(A,B,C,0,0,n,m) où C est un tableau ne contenant initialement aucun élément (**taille**(C)=0).

H. Mraoui

76

## La récursivité : Exercices

### • Exercice 2 :

Nous voulons un sous-algorithme récursif qui permet de trier un tableau. Son principe est de parcourir le tableau **T** en la divisant systématiquement en deux sous-tableaux **T1** et **T2**. L'un est tel que tous ses éléments sont inférieurs à tous ceux de l'autre tableau et en travaillant séparément sur chacun des deux sous-tableaux en réappliquant la même division à chacun des deux sous-tableaux jusqu'à obtenir uniquement des sous-tableaux à un seul élément.

## La récursivité : Exercices

### • **Pour partitionner un tableau en deux sous-tableaux **T1** et **T2** :**

- on choisit une valeur quelconque dans le tableau **L** (la dernière par exemple) que l'on dénomme **pivot**,
- puis on construit le sous-tableau **T1** comme comprenant tous les éléments de **T** dont la valeur est inférieure ou égale au **pivot**,
- et l'on construit le sous-tableau **T2** comme constitué de tous les éléments dont la valeur est supérieure au **pivot**.



### La récursivité : Exercices

- $T = [4, 23, 3, 42, 2, 12, 45, 18, 38, 15]$   
prenons comme **pivot** la dernière valeur  
**pivot = 15**
- Nous obtenons par exemple :  
 $T1 = [4, 12, 3, 2]$   
 $T2 = [23, 45, 18, 38, 42]$
- A cette étape voici l'arrangement de **T** :  
 $T = T1 + \text{pivot} + T2 =$   
 $[4, 12, 3, 2, 15, 23, 45, 18, 38, 42]$

### La récursivité : Exercices

- Il est proposé de **choisir arbitrairement le pivot** que l'on cherche à placer.
- Si le tableau est de longueur nulle, il n'y a rien à faire.
- Sinon, on parcourt le tableau, une fois de gauche à droite, et une autre de droite à gauche, à la recherche d'éléments mal placés, que l'on permute. Si les deux parcours se croisent, on arrête.

## La récursivité : Exercices

***Appliquons cette démarche à l'exemple précédent :***

**T** = [ 4, 23, 3, 42, 2, 12, 45, 18, 38, 15 ]

Choix arbitraire du **pivot** : l'élément le plus à droite ici **15**

## La récursivité : Exercices

- Balayage à gauche :
  - - 4** < **15** => il est dans la bonne sous-liste, on continue
  - liste en cours de construction : [ **4,15** ]
  - **23** > **15** => il est mal placé il n'est pas dans la bonne sous-liste, on arrête le balayage gauche,
  - liste en cours de construction : [ **4,23, 15** ]
  - Balayage à droite :
  - - 38** > **15** => il est dans la bonne sous-liste, on continue
  - liste en cours de construction : [ **4,23, 15, 38** ]
  - **18** > **15** => il est dans la bonne sous-liste, on continue
  - liste en cours de construction : [ **4,23, 15, 18, 38** ]
  - **45** > **15** => il est dans la bonne sous-liste, on continue
- liste en cours de construction : [ **4,23, 15, 45, 18, 38** ]

### La récursivité : Exercices

- $12 < 15 \Rightarrow$  il est mal placé il n'est pas dans la bonne sous-liste, on arrête le balayage droit,
- liste en cours de construction : [ 4, 23, 15, 12, 45, 18, 38 ]
- Echange des deux éléments mal placés :
- [ 4, 23, 15, 12, 45, 18, 38 ] ----> [ 4, 12, 15, 23, 45, 18, 38 ]  
On reprend le balayage gauche à l'endroit où l'on s'était arrêté :  
[ 4, 12, 3, 42, 2, 23, 45, 18, 38, 15 ]
- $3 < 15 \Rightarrow$  il est dans la bonne sous-liste, on continue
- liste en cours de construction : [ 4, 12, 3, 15, 23, 45, 18, 38 ]
- $42 > 15 \Rightarrow$  il est mal placé il n'est pas dans la bonne sous-liste, on arrête de nouveau le balayage gauche,
- liste en cours de construction : [ 4, 12, 3, 42, 15, 23, 45, 18, 38 ]
- On reprend le balayage droit à l'endroit où l'on s'était arrêté :  
[ 4, 12, 3, 42, 2, 23, 45, 18, 38, 15 ]

### La récursivité : Exercices

- $2 < 15 \Rightarrow$  il est mal placé il n'est pas dans la bonne sous-liste, on arrête le balayage droit,
- liste en cours de construction :  
[ 4, 12, 3, 42, 15, 2, 23, 45, 18, 38 ]
- On procède à l'échange :
- [ 4, 12, 3, 2, 15, 42, 23, 45, 18, 38 ]

Donc ;

le pivot : 16

- la sous-liste de gauche :  $T1 = [4, 12, 3, 2]$
- la sous-liste de droite :  $T2 = [23, 45, 18, 38, 42]$
- la liste réarrangée : [ 4, 12, 3, 2, 15, 42, 23, 45, 18, 38 ]

## La récursivité : Exercices

**Procédure** **echanger**(T : tableau[ ] d'entiers, a: Entier, b: Entier)

**Variable**    temp : Entier

**DébutProcédure**

temp  $\leftarrow$  T[a]

T[a]  $\leftarrow$  T[b]

T[b]  $\leftarrow$  temp

**FinProcédure**

## La récursivité : Exercices

**Procédure** **TriRapide** (T : tableau[ ] d'entiers, g: Entier, d: Entier)

**Variable**    i,j,pivot : Entier

**DébutProcédure**

i  $\leftarrow$  g

j  $\leftarrow$  d

pivot  $\leftarrow$  T[g]

## La récursivité : Exercices

Faire

```
Tant que  $T[i] < \text{pivot}$  Faire  $i \leftarrow i+1$  FinTantque
Tant que  $T[j] > \text{pivot}$  Faire  $j \leftarrow j-1$  FinTantque
  Si  $(i \leq j)$  Alors
     $\text{echanger}(T, i, j)$ 
     $i \leftarrow i+1$ 
     $j \leftarrow j-1$ 
  FinSi
Jusqu'à  $(i > j)$ 
```

## La récursivité : Exercices

```
Si  $(g < j)$  Alors
   $\text{TriRapide}(T, g, j)$ 
Finsi

Si  $(i < d)$  Alors
   $\text{TriRapide}(T, i, d)$ 
Finsi
FinProcédure
```

### La récursivité Terminale : Exemple 1

**Fonction** **Fact\_T** ( **n** : Entier, **k** : Entier ) : Entier

/\* le nombre **k** est un accumulateur \*/

- **DébutFonction**

**Si** ( **n** = 0 ) **Alors**

Retourne **k**

**Sinon**

**Fact\_T**( **n**-1, **n**\***k** )

**Finsi**

- **FinFonction**

/\* Appel initial est **Fact\_T**(**n**, 1) \*/

H. Miraoui

89

### La récursivité Terminale : Exemple 2

**Fonction** **Fib\_T** ( **n** : Entier, **a** : Entier, **b** : Entier ) : Entier

- **DébutFonction**

**Si** ( **n** = 0 ) **Alors**

Retourne **a**

**Sinon**

Retourne **Fib\_T**(**n**-1,**b**,**a**+**b**)

**Finsi**

- **FinFonction**

/\* Appel initial est **Fib\_T**(**n**,0,1) \*/

H. Miraoui

90

## Structures et enregistrements

La faculté des sciences d'oujda organise les informations concernant la filière SMI-S3 dans une liste identique à la suivante :

Numéro d'examen	Nom & Prénom	Moyenne	Observation
1	Idrissi Hassan	12	Validé
2	Salhi Yahia	08	Non Validé
.....	.....	.....	.....

## Structures et enregistrements

**Problème :** Le chef de filière veut créer un programme permettant la saisie et le traitement de cette liste sachant qu'elle comporte au maximum 200 étudiants.

- ✚ Donnez la structure de données nécessaire pour les objets à utiliser.
- ✚ Donnez une déclaration algorithmique de ces objets.

## Structures et enregistrements

Objet	Type / Nature	Rôle
Num	Tableau de 200 entiers	Tableau des numéros des étudiants
Nom	Tableau de 200 chaînes	Tableau contenant les noms & prénoms
Moy	Tableau de 200 réels	Tableau des moyennes
Obser	Tableau de 200 chaînes	Tableau des observations

## Structures et enregistrements

**Problème :** Cette approche est totalement ingérable dès qu'il s'agit de :

- + trier les moyennes.
- + rechercher la liste des étudiants ayant validé le semestre
- + rechercher la liste des étudiants autorisés à passer l'examen de rattrapage.

**Cela devient difficile.**



## Structures et enregistrements

Pour proposer une solution pratique, il faudrait une sorte de type particulier qui pourrait regrouper en une seule liste des variables de types différents. Ces types existent. Ils s'appellent des **structures** et permettent de décrire des **enregistrement**. les enregistrements sont des structures de données dont les éléments peuvent être de type différent et qui se rapportent à la même entité .

Les éléments qui composent un enregistrement sont appelés **champs**.

## Déclaration : type structuré

- ✚ Avant de déclarer une variable structure, il faut avoir au préalable défini son type, c'est à dire le nom et le type
- ✚ Un type structuré doit être déclaré et défini avant les variables pour qu'il puisse être utilisé pour définir des variables de type structuré.
- ✚ Vous devez déclarer les types structurés hors de l'algorithme et des sous-algorithmes.

**Définition :** Une **structure** est un type de données défini par l'utilisateur et qui permet de grouper un nombre fini d'éléments (ou **champs**) de types éventuellement différents.

Type

**Structure** nom\_type

champ1: type\_champ1

.....

champn: type\_champn

FinStructure

✚ Chaque **structure** porte un **nom**. Ce nom sera utilisé pour déclarer des **enregistrements**.

✚ Une **structure** peut contenir 1 à n **champs**, du même type ou de types différents.

✚ Une **structure** a un seul **champ** est en soi totalement inutile.

## Déclaration : type structuré

### Exemple :

**Type**

**Structure** tetudiant

**Num:** Entier

**Nom :** chaîne de caractères

**Moy :** Réel

**Oberv :** chaîne de caractères

**FinStructure**

## Déclaration : type structuré

Une fois qu'on a défini un **type structuré**, on peut déclarer des variables **enregistrements** exactement de la même façon que l'on déclare des variables d'un **type primitif**.

### Syntaxe :

**Variable**

**nom\_var :** **nom\_type**

### Syntaxe :

**Variable**

**étudiant1, étudiant2 :** **tétudiant**

### Accès aux champs d'un enregistrement

---

Alors que les éléments d'un tableau sont accessibles au travers de leur indice, les **champs** d'un **enregistrement** sont accessibles à travers leur nom, grâce à l'opérateur '.', i.e., vous accédez aux champs d'un enregistrement en passant par le nom de l'enregistrement et le nom du champ séparé par le caractère '.', le point, selon la forme suivant :

**nom\_enreg.nom\_champ**

### Accès aux champs d'un enregistrement

---

Reprenons l'exemple précédent :

étudiant.Num ← 1

étudiant.Nom ← Idrissi Hassan

étudiant.Moy ← 15

étudiant.Obser ← Validé

## Manipulation d'un enregistrement

Les **champs** d'un **enregistrement** se manipulent exactement comme des **variables**. Ils peuvent recevoir des valeurs et leur valeur peut être affectée à une autre variable. Les **champs** peuvent être utilisés partout où les variables sont utilisées, y compris comme **paramètres** de de sous-programmes, en saisie, en affichage, etc.

## Manipulation d'un enregistrement

**Exemple :** Prendre un catalogue de produits dans un magasin. Un article est décrit par une référence, un nom et un prix.

Type

Structure tarticle

ref : chaîne de caractère

nom : chaîne de caractère

prix : Réel

FinStructure

## Manipulation d'un enregistrement

### Variable

article1, article2 : tarticle

reponse : booléen

### Début

Afficher (" Référence du premier article ? ")

Lire (article1.ref)

Afficher (" Nom du premier article ? ")

Lire (article1.nom)

## Manipulation d'un enregistrement

Afficher (" Prix du premier article ? ")

Lire (article1.prix)

Afficher (article1.ref, article1.nom, article1.prix)

Afficher (" Copier le premier article dans le second ? ")

Lire (reponse)

Si reponse= "oui " Alors

article2 ← article1

FinSi

## Manipulation d'un enregistrement

```
article2.prix ← 15.25
  Si article2.prix = article1.prix Alors
    Afficher (" Les deux articles ont le même prix ")
  FinSi
Fin
```

## L'imbrication d'enregistrements

**Un type structuré peut être utilisé comme type pour des champs d'un autre type structuré.**

### Exemple :

Type

Structure tfabricant

ref : chaîne de caractère

nom : chaîne de caractère

tel : chaîne de caractère

FinStructure

**Un type structuré peut être utilisé comme type pour des champs d'un autre type structuré.**

Type

Structure tarticle

ref : chaîne de caractère

nom\_art : chaîne de caractère

prix : Réel

**fab : tfabricant**

FinStructure

Maintenant déclarez un enregistrement de type

article : Variable article1 : tarticle

Pour accéder aux champs de l'enregistrement article1, il faut utiliser :

article1.ref ← "article11\_11 "

article1.nom ← "ABC "

article1.prix ← "200 "

article1.fab.ref ← "Fab110 "

article1.fab.nom ← "Nom\_fab\_art "

article1.fab.tel ← "043534 "



## Tableau dans une structure

On peut ajouter un tableau comme champ de structure :

```
Type
Structure nom_type
.....
Champ_t : tableau[0..nbelement-1] de type
.....
FinStructure
```

Pour accéder au tableau : `nom_enreg. Champ_t[indice]`

## Tableau dans une structure

**Exemple :** Nous voulons connaître le nombre d'articles vendus sur les 12 mois de l'année.

```
Type
Structure bilanart
    art : tarticle
    vente : tableau[0..11] de réels
FinStructure
```

## Tableau dans une structure

**Déclarer** `bart: bilanart`

Donc

```
Pour i allant de 1 jusqu'à 12 Faire
  Afficher (" Vente du mois ",i, " ?")
  Lire (bart.vente[i-1])
  total ← total+bart.vente[i]
FinPour
```

## Passage d'un enregistrement en paramètre

Il est possible de **passer** tout un **enregistrement** en **paramètre** d'une **fonction** ou d'une **procédure** (on n'est pas obligé de passer tous les **champs** uns à uns, ce qui permet de diminuer le nombre de paramètres à passer), exactement comme pour les **tableaux**.

```
Procédure Proc_afficher (article1 : tarticle)
DébutProcédure
  Afficher (article1.ref, article1.nom, article1.prix)
FinProcédure
```

## Les tableaux d'enregistrement (ou tables)

Il arrive souvent que l'on veuille traiter non pas un seul **enregistrement** mais plusieurs. Par exemple, on veut pouvoir représenter plusieurs articles. On va créer un tableau regroupant toutes les articles d'un catalogue. Il s'agit alors d'un **tableau d'enregistrements**.

## Les tableaux d'enregistrement (ou tables)

Soit la structure **tarticle** :

Type

Structure tarticle

ref : chaîne de caractère

nom : chaîne de caractère

prix : Réel

FinStructure

Vous voulez créer une table de dix articles

**Variable** **articles** : tableau[0..9] de **tarticles**

## Les tableaux d'enregistrement (ou tables)

Chaque élément du **tableau** est un **enregistrement**, contenant plusieurs variables de type différent. On accède à un **enregistrement** par son **indice** dans le **tableau**.

- **articles[1]** représente le deuxième article du catalogue
- **articles[1].nom** représente le nom de deuxième article du catalogue

## Les tableaux d'enregistrement (ou tables)

Pour accéder aux dix enregistrements, le mieux est d'utiliser une boucle :

**Début**

**Pour i allant de 1 jusqu'à 10 Faire**

**Afficher** (" Saisir ref article ",i)

**Lire** (articles[i-1].ref)

**FinPour**

**Fin**

### Définition :

Un fichier est un ensemble structuré de données de même type, nommé et enregistré sur un support lisible par l'ordinateur (disque dur, disquette, flash disque, CD Rom, ..etc). Un fichier peut contenir des caractères (fichier textes), des programmes, des valeurs (fichier de données).

### Organisation des fichiers :

Un fichier se distingue des autres par quelques attributs dont son nom et sa catégorie. Ils se distinguent aussi entre eux par l'organisation de leurs données ce qui définit leur format.

**Catégories de fichiers :** Deux catégories de fichiers sont distinguables :

- ✚ **Les fichiers organisés sous forme de lignes de texte successives, qui s'appellent des fichiers texte. Cela signifie vraisemblablement que ce fichier contient le même genre d'information à chaque ligne. Ces lignes sont alors appelées des enregistrements.**

**Catégories de fichiers :** Deux catégories de fichiers sont distinguables :

- ✚ **Le second type de fichier : il rassemble les fichiers qui ne possèdent pas de structure de lignes (d'enregistrement). Ces fichiers contenant des données variées dont des nombres représentés sous forme binaire. Ces fichiers sont appelés des fichiers binaires**

**Structure des enregistrements:** les fichiers peuvent être structurés en enregistrement. Il y a deux grandes possibilités pour structurer ces enregistrements :

✚ La structure n°1 est dite **délimitée** ; Elle utilise un **caractère spécial**, appelé caractère de **séparation** ou **délimitation**, qui permet de repérer quand finit un **champ** et quand commence le suivant.

**Exemple :**  
**Structure n°1**

```
"Idrissi1";"Hassane1";0123;"Hassane1@yahoo.fr"  
"Idrissi2";"Hassane2";0456;"Hassane2@yahoo.fr"  
"Idrissi3";"Hassane3";0789;"Hassane3@yahoo.fr"  
"Idrissi4";"Hassane4";0978;"Hassane4@yahoo.fr"
```

**Remarque :** Le caractère de délimitation ne doit pas se retrouver à l'intérieur de chaque champ.

✚ **La structure n°2** est dite à **champs de largeur fixe**, il n'y a pas de **délimiteurs**.

**Chaque champ a une longueur prédéfinie et occupe toute cette longueur, quitte à être complété par des espaces.**

Idrissi1	Hassane1	0123	Hassane1@yahoo.fr
Idrissi2	Hassane2	0456	Hassane2@yahoo.fr
Idrissi3	Hassane3	0789	Hassane3@yahoo.fr
Idrissi4	Hassane4	0978	Hassane4@yahoo.fr

**Remarque :** Contrairement au format limité, le format à largeur fixe consomme bien plus de mémoire. Cependant, la récupération de tels champs est bien plus simple car vous connaissez à l'avance la taille de chaque champ et donc toutes les positions pour découper vos enregistrements.



**Types d'accès :**

Le type d'accès est la manière dont la machine va pouvoir aller rechercher les informations contenues dans le fichier.

On distingue :

- **L'accès séquentiel** : Pour lire une information particulière, il faut lire toutes les informations situées avant.

**Types d'accès :**

- **L'accès direct** : Nous pouvons accéder directement à l'information désirée, en précisant le numéro d'emplacement (le numéro d'ordre) de cette information.

### Les fichiers à accès séquentiel

Pour travailler avec des fichiers, vous devez respecter un certain ordre. Il vous faudra :

- ✚ **Ouvrir le fichier** : c'est-à-dire indiquer à quel fichier vous voulez accéder.
- ✚ **Traiter le contenu du fichier** : le lire, y écrire bref toutes les opérations désirées et manipuler son contenu.
- ✚ **Fermer le fichier** : quand tous les traitements sont terminés

Remarque : Si l'on veut travailler sur un **fichier**, la première chose à faire est de l'**ouvrir**. Cela se fait en attribuant au **fichier** un **numéro de canal**. On ne peut ouvrir qu'un seul fichier par canal, mais quel que soit le langage, on dispose toujours de **plusieurs canaux**. Donc, l'accès à un fichier passe par l'utilisation d'un canal. Un **canal** permet de faire transiter un flux d'informations d'un programme vers un **fichier**.

## Les Fichiers

---

Les modes d'ouverture L'important est que lorsqu'on ouvre un fichier, on souhaite **lire** son contenu, y **écrire**, ou **ajouter** des lignes à la fin.

✚ **Si** on ouvre un fichier **pour lecture**, on pourra uniquement récupérer les informations qu'il contient, sans les modifier en aucune manière.

## Les Fichiers

---

✚ **Si** on ouvre un fichier pour **écriture**, on pourra mettre dedans toutes les informations que l'on veut. Mais les informations précédentes, si elles existent, seront **intégralement écrasées**. Et on ne pourra pas accéder aux informations qui existaient précédemment.

✚ Si on ouvre un fichier **pour ajout**, on ne peut **ni lire**, **ni modifier** les informations existantes. Mais on pourra, comme vous commencez à vous en douter, ajouter de nouvelles lignes.

### Déclaration

La structure fichier se déclare comme un **type prédéfini** :

nom\_fichier : **fichier séquentiel**

Pour ouvrir un fichier texte, on écrira par exemple :

**Ouvrir** "Toto.txt" dans **nom\_fichier** en **Lecture**

## Les Fichiers

### Algorithme Ouvre

Variable

**fic** : fichier séquentiel

**nom** : chaîne de caractères

- **Début**

**nom** ← "Toto.txt"

Ouvrir **nom** dans **fic** en Lecture

/\* traitements \*/

Fermer **fic**

- **Fin**

## Les Fichiers

### Lire des enregistrements (lignes)

La lecture d'une ligne se fait via l'instruction **Lire**. **Lire** lit l'enregistrement présent à la position actuelle du fichier, puis se place sur l'enregistrement suivant.

### La syntaxe est la suivante

**Lire**(nom\_fichier, variable)

## Les Fichiers

### Algorithme Lire\_fichier

Variable

**fic** : fichier séquentiel

**Ligne, Nom, Prénom, Tel, Mail**: chaîne de caractères

- **Début**

Ouvrir "Toto.txt" dans **fic** en Lecture

Lire (**fic**, **Ligne**)

**Nom** ← Milieu(**Ligne**, 1, 20)

**Prénom** ← Milieu(**Ligne**, 21, 15)

**Tel** ← Milieu(**Ligne**, 36, 10)

**Mail** ← Milieu(**Ligne**, 46, 20)

Fermer **fic**

- **Fin**

H. Mraoui

137

## Les Fichiers

Lire un fichier séquentiel de bout en bout suppose de programmer une **boucle**.

Comme on sait rarement à l'avance combien d'enregistrements comporte le fichier, la combine consiste à utiliser la fonction **FinFichier()**. Cette fonction prend en paramètre le nom du fichier. Elle retourne **Vrai** si la fin du fichier a été atteinte.

H. Mraoui

138

## Les Fichiers

### Algorithme Lire\_fichier

Variable

**fic** : fichier séquentiel

**Ligne**: chaîne de caractères

**i**: Entier

**Nom, Prénom, Tel, Mail**: tableau[ ] de chaîne de caractères

- **Début**

Ouvrir "Toto.txt" dans **fic** en Lecture

**i** ← -1

## Les Fichiers

**Tantque** Non **FinFichier**(**fic**) **Faire**

**Lire** (**fic**,**Ligne**)

**i** ← **i**+1

**Taille**(**Nom**) ← **Taille**(**Nom**)+1

**Taille**(**Prénom**) ← **Taille**(**Prénom**)+1

**Taille**(**Tel**) ← **Taille**(**Tel**)+1

**Taille**(**Mail**) ← **Taille**(**Mail**)+1

**Nom**[**i**] ← **Milieu**(**Ligne**, 1, 20)

**Prénom**[**i**] ← **Milieu**(**Ligne**, 21, 15)

**Tel**[**i**] ← **Milieu**(**Ligne**, 36, 10)

**Mail**[**i**] ← **Milieu**(**Ligne**, 46, 20)

**FinTantque**

Fermer **fic**

- **Fin**

## Les Fichiers

### Algorithme Lire\_fichier

#### Type

**Structure** Personne

**Nom, Prénom, Tel, Mail:** chaîne de caractères

**FinStructure**

#### Variable

**fic** : fichier séquentiel

**Ligne:** chaîne de caractères

**i** : Entier

**Base** : tableau[ ] de **Personne**

H. Mraoui

141

## Les Fichiers

### Début

**Ouvrir** "Toto.txt" dans **fic** en **Lecture**

**i** ← -1

**Tantque** Non **FinFichier(fic)** **Faire**

**i** ← **i**+1

**Taille(Base)** ← **Taille(Base)**+1

**Lire (fic,Base[i])**

**FinTantque**

**Fermer fic**

### Fin

H. Mraoui

142



### Ecrire

L'écriture utilise l'instruction **Ecrire**, une fonction qui prend comme paramètre le nom du fichier et l'enregistrement (la ligne).

### La syntaxe est la suivante

**Ecrire**(nom\_fichier, ligne)

### Ajouter

**Ajout** utilise l'instruction **Ecrire**, une fonction qui prend comme paramètre le nom du fichier et l'enregistrement (la ligne).

### La syntaxe est la suivante

**Ecrire**(nom\_fichier, ligne)

Avec

**Ouvrir** "Toto.txt" dans **nom\_fichier** en **Ajout**

## La complexité

---

- Quand on tente à résoudre un problème, la question qui se pose c'est le choix du **meilleur algorithme** parmi les algorithmes qui permettent de décrire des méthodes de résolution de ce problème.
- Certains algorithmes sont complexes et le traitement peut nécessiter beaucoup de **temps** et de **ressources de machine**, c'est qu'on appelle le "**coût** " (**efficacité** ou **complexité**) de l'algorithme.

## La complexité

---

- L'analyse de la **complexité** des algorithmes étudie formellement la **quantité de ressources** en **temps** et en **espace** nécessitée par l'exécution d'un **algorithme** donnée.

## La complexité

---

Le temps d'exécution dépend de

- ✚ Le problème à résoudre
- ✚ La taille des données
- ✚ L'algorithme de résolution
- ✚ L'expertise du programmeur
- ✚ L'habilité du programmeur
- ✚ La rapidité de la machine
- ✚ Le langage de programmation
- ✚ Le compilateur

H. Mraoui

147

## La complexité

---

- Dans ce cours nous intéressons au coût des actions résultant de l'exécution d'un algorithme, en fonction de **la taille des données** traitées.
- Ceci nous permet de **comparer des algorithmes** traitant le même algorithme.

H. Mraoui

148

### Définition (Complexité)

La **complexité** d'un algorithme désigne le nombre d'opérations fondamentales ( affectation, comparaison, opérations arithmétiques, ...).

*La **complexité** s'exprime en fonction de la taille  $n$  des données.*

### Mesure de la complexité

On s'intéresse généralement

- ✚ **Meilleur cas** (cas favorable).
- ✚ **Pire cas** (cas défavorable).
- ✚ **Cas moyen** (complexité moyen pour toutes les entrées possibles).

## La complexité

Par exemple pour la recherche d'un élément dans une liste :

12	32	22	10	5	45
----	----	----	----	---	----

- ✚ **Le meilleur cas** est que l'on trouve l'élément à la première comparaison.
- ✚ **Le pire cas** est qu'on parcourt tous les éléments de la liste et que l'élément recherché ne s'y trouve pas.

## La complexité

- ✚ **Le moyen cas** est que l'élément recherché se trouve à la 2<sup>ème</sup>, 3<sup>ème</sup> position par exemple

**Remarque** : Dans la suite nous intéressons particulièrement au pire cas.

### Mesure de la complexité

Le choix de l'**unité** de mesure ne dépend pas de la nature précise des données mais de leur taille  $n$ .

- On désigne par  $C(n)$  le nombre d'opérations effectuées pour exécuter un algorithme donné dont la taille de données est  $n$ .
- $C(n)=O(f(n))$  : " Complexité en  $f(n)$  "  
Généralement la fonction  $f$  est une combinaison de polynômes, logarithmes, ou exponentielle

- $C(n)=O(f(n))$  : " Complexité en  $f(n)$  "  
signifie que le nombre d'opérations effectuées est borné  $K*f(n)$  lorsque  $n$  tend vers l'infini, c'est-à-dire  
Il existe  $K>0$  et  $n_0$  telles que pour tout  $n > n_0$ ,  
 $C(n) < K*f(n)$ .

### Pourquoi étudier le comportement à l'infini

On se préoccupe surtout de la croissance de la complexité en fonction de la taille des données.

### Classes de complexité

- ✚  $O(1)$  : complexité constante.
- ✚  $O(\log(n))$  : complexité logarithmique.
- ✚  $O(n)$  : complexité linéaire.
- ✚  $O(n \cdot \log(n))$  : complexité quasi-linéaire.
- ✚  $O(n^2)$  : complexité quadratique.
- ✚  $O(n^p)$  : complexité polynomiale.
- ✚  $O(2^n)$  : complexité exponentielle.

$O(n!)$  : complexité factorielle.

Sur Intel Pentium 4 à 3.2 GHz, si vous traitez 20 données dans un algorithme de complexité  $O(n!)$  le temps d'exécution est autour de 25 ans.



### Calcul de la complexité

- Opération de base comme par exemple affectation, lecture, écriture,.. la complexité dans ce cas est  $O(1)$ .

- Instruction séquentielle

$$C(I_1; \dots; I_p) = \max(C(I_1), \dots, C(I_p))$$

- Instructions conditionnelles

$$C(\text{si } \textcolor{red}{cond} \text{ alors } I_1 \text{ sinon } I_2) =$$

$$\textcolor{red}{C}(cond) + \max(\textcolor{red}{C}(I_1), \textcolor{red}{C}(I_2))$$

### Calcul de la complexité

- Instruction itératives

$$\textcolor{red}{C}(\text{pour } i = i_1 \text{ jusqu'à } i_2 \text{ faire } T_i) = \\ (i_2 - i_1 + 1) * \max(\textcolor{red}{C}(T_i)).$$

### Calcul de la complexité

---

Exemple : Exprimer la complexité de cet Algorithme :

```
fact ← 1           // Initialisation
Tant que (n>1) Faire
    fact ← fact*n
    n ← n - 1
Fin tant que
```

### Calcul de la complexité

---

Exemple : Exprimer la complexité des algorithmes qui permettent de chercher un élément dans un tableau.

## Calcul de la complexité

**Fonction R1**(T : tableau[ ] d'entiers, x: Entier) : booléen

**Variable**

i: Entier

• **DébutFonction**

• i ← 1

• **Faire**

• **Si** ( T[i]=x)

• **Retourner Vrai**

• **Sinon**

• i ← i+1

• **FinSi**

• **Jusqu'à** (i>=Taille(T)-1)

• **Retourner Faux**

• **FinFonction**

$$C(n) = O(n^2)$$

## Calcul de la complexité

**Fonction R2**(T : tableau[ ] d'entiers, x: Entier) : booléen

**Variable**

i,k: Entier

• **DébutFonction**

• i ← 1

• k ← Taille(T)

• **Pour** i allant de 0 à k-1 **Faire**

• **Si** ( T[i]=x)

• **Retourner Vrai**

• **FinSi**

• **Fin Pour**

• **Retourner Faux**

• **FinFonction**

$$C(n) = O(n)$$

### Comment calculer la complexité d'un algorithme ?

- ✚ Si  $C(n+1)=C(n)$  alors  $C(n)=O(1)$ .
- ✚ Si  $C(2n)=C(n)+1$  alors  $C(n)=O(\log(n))$ .
- ✚ Si  $C(n+1)=C(n)+1$  alors  $C(n)=O(n)$ .
- ✚  $C(n) = 2 * C(n/2) + n$  alors  $C(n)=O(n \times \log(n))$
- ✚ Si  $C(n+1)=C(n)+n$  alors  $C(n)=O(n^2)$ .
- ✚ Si  $C(n+1)=2*C(n)$  alors  $C(n)=O(2^n)$ .

### Evaluation de $C(n)$ (Fonctions récursives)

Fonction **FunctionRecursive** ( $n$  : Entier)

1. Si ( $n > 1$ ) alors
2. **FunctionRecursive**( $n/2$ ), coût  $C(n/2)$
3. **Traitement**( $n$ ), coût  $T(n)$
4. **FunctionRecursive**( $n/2$ ), coût  $C(n/2)$

Equation récursive :

$$C(n) = 2 * C(n/2) + T(n)$$

Si  $T(n) = 1$  alors  $C(n) = O(n)$

Si  $T(n) = n$  alors  $C(n) = O(n \times \log n)$

### Evaluation de $C(n)$ (Fonctions récursives)

– **Problème :** calculer  $x^n$

**données :**  $x$  : réel ,  $n$  : entier

*Méthode 1 :*  $x^0 = 1$ ;  $x^i = x * x^{i-1}$   $i > 0$

*Méthode 2 :*  $x^0 = 1$ ;

$x^i = x^{i/2} * x^{i/2}$  , si  $i$  est pair;

$x^i = x * x^{i/2} * x^{i/2}$  si  $i$  est impair

...

**résultats :**  $y = x^n$

– Laquelle choisir? et pourquoi?

### Evaluation de $C(n)$ (Fonctions récursives)

#### Recherche dichotomique :

On suppose qu'on dispose d'un tableau et on se donne un élément quelconque et on cherche si cet élément est dans le tableau ou non.

**Attention :** uniquement si le tableau est déjà trié.

**Question :** Exprimer la complexité de la recherche dichotomique.

## Evaluation de $C(n)$ (Fonctions récursives)

**Fonction** Recherche\_Dicho(T : tableau[ ] d'entiers, ind\_min: Entier, ind\_max : Entier, x: Entier) : booléen

**Variable**

ind\_moyen : Entier                      /\* Le tableau est trié dans un ordre croissant \*/

• **DébutFonction**

```

•       Si (ind_min<=ind_max) alors
•           ind_moyen ← (ind_min+ind_max) div 2
•           Si ( T[ind_moyen]=x) Alors
•               retourne Vrai
•           Sinon
•               Si ( T[ind_moyen]>x) Alors
•                   retourne Recherche_Dicho(T,ind_min,ind_moy-1,x)
•               Sinon
•                   retourne Recherche_Dicho(T,ind_moy+1,ind_max,x)
•               Finsi
•           Finsi
•       Sinon
•           retourne Faux
•       Finsi FinFonction
    
```

H. Mraoui

169

## Evaluation de $C(n)$ (Fonctions récursives)

### Recherche dichotomique :

A Chaque appel, on divise le tableau de recherche en 2. Soit  $k$  tel que  $n=2^k$

$$\begin{aligned}
 \text{Donc} \quad C(n) &= 1 + C(n/2) \\
 &= 1 + C(2^{k-1}) \\
 &= 2 + C(2^{k-2}) \\
 &= \dots \\
 &= k + C(1) = \log_2(n) + C(1)
 \end{aligned}$$

**Donc la complexité est en  $\log_2(n)$ .**

H. Mraoui

170

### Evaluation de $C(n)$ (Fonctions récursives)

**Procédure TriRapide** (T : tableau[ ] d'entiers, g: Entier, d: Entier)

**Variable** i,j,pivot : Entier **DébutProcédure** i ← g j ← d

ind\_moyen ← (g+d) div 2                      pivot ← T[ind\_moyen]

**Faire**

Tant que T[i] < pivot **Faire** i ← i+1 **FinTantque**

Tant que T[j] > pivot **Faire** j ← j-1 **FinTantque**

**Si** (i <= j) **Alors** echanger(T, i, j) i ← i+1 j ← j-1 **FinSi**

**Jusqu'à** (i > j)

**Si** (g < j) **Alors** TriRapide(T, g, j) **Finsi**

**Si** (i < d) **Alors** TriRapide(T, i, d)

**Finsi**

**FinProcédure**

**Question** : Exprimer la complexité de cette  
procédure.

H. Mraoui

171

### Evaluation de $C(n)$ (Fonctions récursives)

Si on prend  $\text{pivot} = T[p]$  avec  $0 \leq p \leq n-1$ .

La taille des données est évidemment le cardinal  $n$  du tableau à trier. La formule de récurrence suivante somme respectivement le coût du calcul du partitionnement et du test, soit  $O(1) + O(n) = O(n)$  et des deux appels récursifs :

$$C(n) = O(n) + C(p+1) + C(n - p - 1)$$

avec  $C(1) = O(1)$ .

H. Mraoui

172

### Evaluation de $C(n)$ (Fonctions récursives)

Dans le cas idéal, on a  $p+1=n/2$ , donc

$$C(n)=O(n)+2 \times C(n/2)=O(n \times \log_2(n))$$

Soit  $n=2^k$

$$\begin{aligned}\text{Donc } C(2^k) &= 2^k + 2 \times C(2^{k-1}) \\ &= 2^k + 2 \times (2^{k-1} + 2 \times C(2^{k-2})) \\ &= 2^k + 2^k + 2^2 \times C(2^{k-2}) \\ &= \dots \\ &= k \times 2^k + 2^k \times C(1) \\ &= O(n \times \log_2(n))\end{aligned}$$

### Evaluation de $C(n)$ (Fonctions récursives)

Dans le cas **pire**, on a  $p=0$ , donc

$$C(n)=O(n)+C(1)+C(n-1)=O(n^2)$$

$$\begin{aligned}C(n) &= n + C(1) + C(n-1) \\ &= n + (n-1) + 2 \times C(1) + C(n-2) \\ &= \dots \\ &= O(n^2)\end{aligned}$$



- + Une preuve algorithmique est en général assez délicate.
- + Souvent nos algorithmes sont très simples, et la preuve ressemble à une paraphrase de l'algorithme

Prouver un algorithme est deux choses :

- + Prouver sa **terminaison** : un algorithme doit effectuer un **nombre fini d'étapes** et s'arrêter.
- + Prouver sa **correction** : un algorithme doit faire ce qu'on attend de lui

**Terminaison :**

On dit qu'un algorithme  $P$  termine, si et seulement si, tout état initial  $E$  donne une exécution terminante de  $P$ .

**Terminaison :**

L'algorithme suivant ne termine pas :

**Tant que V Faire**

**Fintantque**

**Terminaison :**

L'algorithme suivant ne termine pas :

```
Si (a<=b) Alors  
    max ← b  
Sinon  
    Tant que (a <> b) faire  
        max ← a  
    FinTantque  
FinSi
```

**Correction :**

Soit l'algorithme ***P*** dont la correction est exprimée par une condition ***C***, On dit que ***P*** est correct si et seulement si, pour tout état initial ***E*** qui donne une exécution terminante

***$F = P(E)$  on a  $F(C) = Vrai.$***

**Invariant :**

Une condition vérifiée sur l'état des données traitées par la récurrence.

**Respect Invariant :**

L'invariant doit être vérifié par chaque étape (appel) de la résolution récursive.

Preuve par invariant de boucle

Montrer que Cet algorithme se termine et en sortie R contient  $x^n$

$A \leftarrow x$     $N \leftarrow n$     $R \leftarrow 1$

**Tant que**  $N > 0$  **faire**

**Si** N pair **alors**

$A \leftarrow A * A$        $N \leftarrow N/2$

**Sinon**

$R \leftarrow R * A$        $N \leftarrow N - 1$

**FinSi**

**FinTantque**