Table des matières

Chapitre1: Introduction à l'algorithmique

I- Les différentes étapes de résolution d'un problème

Pour résoudre un problème en informatique, il faut passer par 5 étapes :

- Définition et analyse du problème
- Ecriture de l'algorithme
- Programmation
- Compilation du programme
- Exécution et test du programme

1. Définition et analyse du problème

Il s'agit de :

- Définir les données qu'on dispose et les objectifs qu'on souhaite atteindre
- Prévoir des réponses à tous les cas envisageables

Exemple : Si le problème est la résolution d'une équation de second degré $ax^2+bx+c=0$

→ Les données sont a, b et c

→ Les sorties sont x_1 et x_2

→ Les cas : $a=0$ et $b\neq0$, $a=0$ et $b=0$, $a\neq0$ ……

2. Ecriture de l'algorithme

C'est la phase la plus difficile et importante, elle fournit la méthode et la démarche que l'ordinateur va suivre pour résoudre le problème posé.

● *Définition d'un algorithme :*

Un algorithme est une séquence d'étapes de calcul qui utilise des données en entrée pour arriver à des résultats en sortie.

3. *Programmation de l'algorithme*

Il s'agit d'exprimer l'algorithme dans un langage connu par l'ordinateur. Il faut donc choisir un langage de programmation et ensuite traduire l'algorithme sous forme d'un programme exprimé dans ce langage.

4. *Compilation*

Il s'agit de traduire le programme écrit dans un langage de haut niveau en un programme exécutable écrit dans un langage binaire de bas niveau tout en détectant les éventuelles erreurs. Cette tâche est réalisée par le compilateur.

5. *Exécution et test du programme*

Il s'agit de s'assurer que le programme donne un résultat correct dans tous les cas et pour toutes les éventualités.

→ Effectuer plusieurs jeux de tests correspondant aux différents cas et vérifier la validité des résultats.

II- Structure générale d'un algorithme

1. *Schéma général d'un algorithme*

Un algorithme comporte généralement deux parties :

- *Partie déclarative* : elle contient l'entête, la déclaration des constantes et celle des variables.
- *Partie corps de l'algorithme* : elle consiste en une séquence d'actions faisant appel à des opérations de base de l'ordinateur.

Syntaxe :

Algorithme « nom de l'algorithme »

Constantes

« Liste des constantes avec leurs valeurs »

Variables

« Liste des variables suivies par leurs types »

Partie déclarative

Début

« Séquence d'actions »

Fin

} **Partie corps de l'algorithme**

Une action peut être :

→ Action d'affectation ou,

→ Action d'entrée- sortie ou,

→ Action de contrôle conditionnelle simple ou à choix multiple ou,

→ Action de répétition.

2. Définition d'une variable

Une variable est un emplacement mémoire capable de contenir des valeurs de type défini au préalable. Elle peut être définie comme une boite qui admet un nom, une taille, un contenu et une adresse.

→ Le nom de la variable s'appelle identificateur de la variable.

→ La taille dépend du type de la variable (exemple : 2 octets pour un entier, 1 octet pour un caractère, 4 octets pour un réel…)

→ L'adresse désigne le numéro du 1er octet occupé par cette variable en mémoire centrale

Dans un algorithme, les variables sont déclarées comme suit :

Variables

Liste des variables suivies par des virgules : type 1

Liste des variables suivies par des virgules : type 2

.

.

Liste des variables suivies par des virgules : type i

→ Dans un algorithme, on peut avoir 0 à plusieurs variables.

Exemple :

Variables

X, Y : entier

A : réel

3. *Définition d'une constante*

La définition d'une constante est la même que celle d'une variable à la différence que la valeur d'une constante reste inchangée tout au long de l'algorithme.

Syntaxe :

Constantes

Nom const1 = val 1: type

Nom consti = val i : type

Exemple:

Constantes

Min = 10 : entier

Max = 200 : entier

4. *Les types de base*

A toute variable est attribué un type qui définit :

- L'ensemble des valeurs que peut prendre la variable
- L'ensemble des opérations qu'on peut appliquer sur la variable

Il existe des types simples qui sont prédéfinis tels que les types : entier, réel, caractère ou booléen.

a) *Type entier*

- Il représente l'ensemble des entiers relatifs tel que : 8, -10, 3……

- Les opérations permises sont : +, -, *, div (division entière) et mod (reste de la division entière)

b) *Type réel*

- Il représente l'ensemble IR

- Deux formes de représentation : La forme usuelle « a.b » exemple : -4.6, 13.9 ….. ou la forme scientifique a E b exemple : 345 = 3.45 E2 = 0.345 E3
- Les opérations permises sont : +, -, *, /

c) Type caractère

- Il peut être une lettre, un chiffre ou caractère spécial exemple : ‘a’, ‘b’, ‘3’ ….
- Les opérations permises : =, ≠, <, <=, >, >=.

d) Type booléen

- Il représente les deux valeurs ‘Vrai’ et ‘Faux’
- Les opérations : NON, ET, OU

Remarque : Il existe des types composés définis à partir des types de base comme les tableaux, les chaînes de caractère….

Chapitre 2 : Les instructions simples

I- Instruction d'affectation

Cette action permet de ranger une nouvelle valeur dans une variable

Syntaxe

Identificateur var ← <expression>

- Expression peut être :
    - Une variable
    - Une constante
    - Une expression arithmétique
    - Une expression logique

Remarque

- Une constante ne peut jamais figurer à gauche d'une affectation.
- Après une affectation, l'ancien contenu est perdu pour être substitué par le nouveau contenu.
- Une action d'affectation doit se faire entre deux types compatibles.

1. Les expressions arithmétiques

<exp-arith> op_arith <exp-arith>

- Op_arith peut être '+', '-', '/' ou '*'

  Exemple : (Y/2) + x*3

- L'ordre de priorité des opérateurs arithmétiques :
    - \- signe négatif
    - ( ) parenthèses
    - ^ puissance
    - * et / multiplication et division
    - + et – addition et soustraction

2. *Les expressions logiques*

- Les expressions logiques admettent Vrai ou Faux comme résultat.
- Elles peuvent utiliser des opérateurs relationnels (= , ≠, <,<=, >, >=) ou des opérateurs logiques (NON, ET, OU)
- L'ordre de priorité est :

NON	>
ET	>=
OU	<
	<=
	=
	≠

Exemple : (x<6) ET (Y = 20) donne vrai si x<6 et Y = 20 et faux sinon

II- Instruction de lecture ou d'entrée

- Elle permet d'affecter, à une variable, une donnée introduite) partir d'une périphérique d'entrée (clavier).
- Syntaxe :

Lire (nom_var1, nom_var2,……)

- Exemple :

Lire(A) : lit une valeur à partir du périphérique d'entrée et la range dans la case mémoire associée à A.

Lire(X,Y) : lit deux valeurs la première pour X et la deuxième pour Y.

III- Instruction d'écriture ou de sortie

- Elle permet d'afficher des résultats sur un périphérique de sortie (écran). Ce résultat peut être :
  - Une chaîne de caractères délimitée par des guillemets " "
  - La valeur d'une variable dont le nom est spécifié

- La valeur d'une expression

Syntaxe :

Ecrire (Liste d'expressions séparées par des virgules)

- L'ordinateur évalue tout d'abord l'expression puis affiche le résultat obtenu

Exemple :

```
Lire (somme)
Lire(Nbre)
Ecrire ("La moyenne est :", somme / Nbre)
```

Si l'utilisateur introduit 120 pour somme et 10 pour Nbre alors l'affichage sera : La moyenne est 12.

- ***Exercice 1 :***

  Ecrire un algorithme qui lit deux entiers X et Y et affiche leurs valeurs avant et après permutation

- ***Exercice 2 :***

  Ecrire un algorithme qui lit trois entiers et qui calcule et affiche leur somme, leur produit et leur moyenne.

Chapitre 3: Les structures conditionnelles

Introduction

En programmation, on est souvent confronté à des situations ou on a besoin de choisir entre 2 ou plusieurs traitements selon la réalisation ou non d'une certaine condition d'ou la notion de traitement conditionnel. On distingue deux structures de traitement conditionnel à savoir :

- La structure conditionnelle simple qui consiste à évaluer une condition (expression logique à valeur vrai ou faux) et d'effectuer le traitement relatif à la valeur de vérité trouvée.
- La structure conditionnelle à choix multiple qui consiste à évaluer une expression qui n'est pas nécessairement à valeur booléenne (elle peut avoir plus de deux valeurs) et selon la valeur trouvée, effectue un traitement.

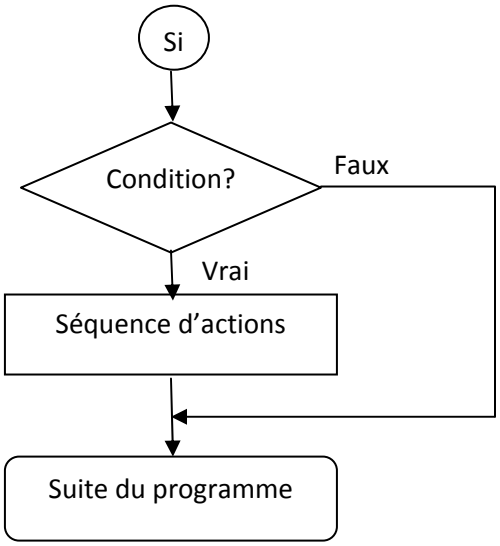
I- Structure conditionnelle simple

1. Forme simple : (Si …. Alors ….. Finsi)

Syntaxe :

Si condition **Alors**
action(s)
Fin si

Dans cette forme, la condition est évaluée. Si elle vaut vrai alors c'est la séquence d'actions qui est exécutée sinon c'est l'action qui suit l'action conditionnelle dans l'algorithme qui est exécutée.
L'exécution de cette instruction se déroule selon l'organigramme suivant :



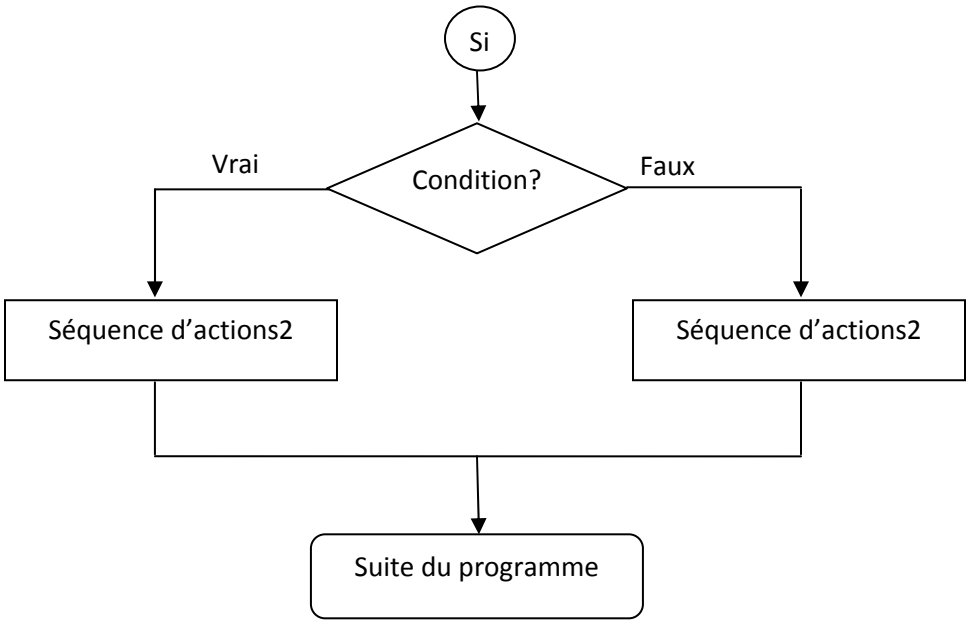
Exemple 1 : Ecrire un algorithme qui permet de saisir un entier et d'afficher impossible d'être diviseur si cet entier est égal à 0.

2. Forme composée : (Si … alors…….sinon)

Syntaxe :

Si condition **Alors**
  Action(s)1
Sinon
  Action(s)2
Fin si

Dans cette forme, la condition est évaluée. Si elle vaut vrai alors c'est la séquence d'actions 1 qui sera exécutée sinon c'est la séquence d'actions 2 qui sera exécutée.
L'exécution de cette instruction se déroule selon l'organigramme suivant :



Exemple 2 :
Ecrire un algorithme qui permet de saisir un entier et d'afficher « pair » si cet entier est pair ou « impair » si cet entier est impair.

3. Forme imbriquée

Syntaxe

Si condition 1 **Alors**
  Action(s)1
Sinon
  Si condition 2 **Alors**
    Action(s)2

Sinon
Si condition N-1 **Alors**
Action(s)N-1
Sinon
Action(s)N
Fin si

Si la condition est vraie, alors la séquence d'actions 1 sera exécutée sinon on évalue la condition 2 si elle est vraie la séquence d'actions 2 sera exécutée. Enfin, si aucune des N-1 conditions est vraie alors on exécute la séquence d'actions N.

Exemple :

Ecrire un algorithme qui permet de saisir deux entiers A et B puis teste si A>B ou A<B ou A=B.

II- Structure conditionnelle à choix multiple

Syntaxe

Selon <sélecteur> faire
<liste de valeurs1> : <traitement 1>
<liste de valeurs2> : <traitement 2>
…..
…..
<liste de valeursN> : <traitement N>
Sinon
<traitement N+1>
Fin selon

- Le sélecteur est un identificateur
- <traitement i> est une séquence d'actions.
- <liste de valeurs i> peut être une constante ou un intervalle de constantes de même type que sélecteur.
- La partie sinon est facultative. Elle est exécutée si aucune des valeurs n'est égale au sélecteur.

Exemple :

Ecrire un algorithme qui permet de lire un numéro de jour de la semaine (compris entre 1 et 7) et d'afficher le nom du jour en toute lettre.

Chapitre 4: Les structures itératives

Introduction

On peut exécuter une action ou un ensemble d'actions non pas infiniment mais un certain nombre de fois : c'est la notion de boucles.

I- La structure « Pour ….. faire…….Finpour »

Syntaxe

Pour vc de vi à vf faire
    Traitement
Finpour

- Vc : compteur de type entier
- Vi et vf : valeur initiale et valeur finale de vc
- Traitement : action ou séquence d'actions à répéter (vf-vi +1) fois.
- La boucle Pour est utilisée lorsque le nombre d'itération est connu à l'avance.
- Vc reçoit une valeur initiale vi pour la première fois, il ne doit pas être modifié par une action de traitement à l'intérieur de la boucle.
- Vc est incrémenté automatiquement par 1 à chaque exécution du corps de la boucle Pour. Cette valeur d'incrémentation est appelée le pas de la boucle.
- L 'exécution de la boucle finit lorsque vc atteind vf.

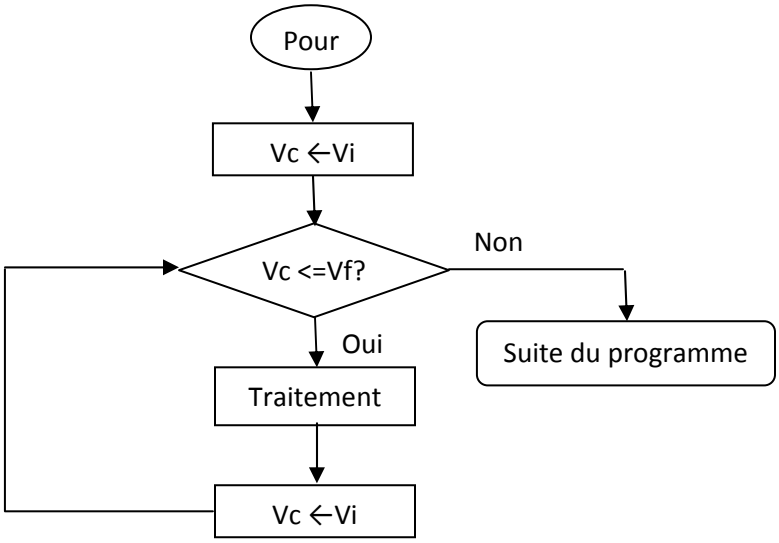


Schéma d'exécution d'une boucle « Pour »

Exemple :
Pour i de 1 à 5 faire
Ecrire (i * 100)
Fin pour
Exécution : i = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6
Résultat 100 200 300 400 500

Remarques :
- Une boucle peut être exécutée une ou plusieurs fois.
- Si le pas est différent de 1, il faut ajouter l'option (pas = constante)

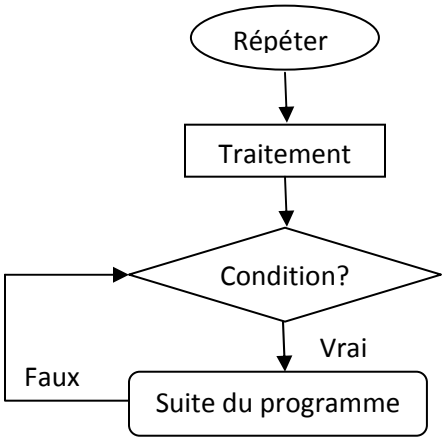
Exemple
Pour i de 5 à 1 (pas = -1) faire
Ecrire (i * 100)
Fin pour
Exécution : i = ~~5~~ ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0
Résultat 500 400 300 200 100

Exemple 1
Ecrire un algorithme qui permet de calculer et d'afficher la somme des nb premiers entiers naturels (nb est saisi à partir de clavier).
Exemple 2
Ecrire un algorithme qui lit un entier n qu'on suppose positif puis affiche tous ses diviseurs.

II- La structure « Répéter …. Jusqu'à »

Syntaxe
Répéter
Traitement
Jusqu'à (condition)

- Condition : condition d'arrêt et de sortie de la boucle
- Traitement : action ou ensemble d'actions à exécuter tant que la condition n'est pas vérifiée, dés qu'elle soit vérifiée, l'exécution du traitement s'arrête.
- Le nombre de répétition n'est pas connu à l'avance.
- Le traitement est exécuté au moins une fois quelque soit le résultat de la condition.
- La condition doit être initialisée avant le début de la boucle et doit être modifiée à l'intérieur de la boucle.



Exemple

i ← 1

Répéter

    Ecrire (i *100)

    i← i +1

jusqu'à (i > 5)

Exécution : i = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

Résultat 100 200 300 400 500

Exemple

Ecrire un algorithme qui permet de calculer la somme des nb premiers entiers en utilisant la boucle répéter jusqu'à

Exemple

Ecrire un algorithme qui permet de calculer la factorielle d'un entier n donné (on suppose que n est un entier positif)

III- La structure « Tantque……. Faire…….. Fintantque »

Syntaxe

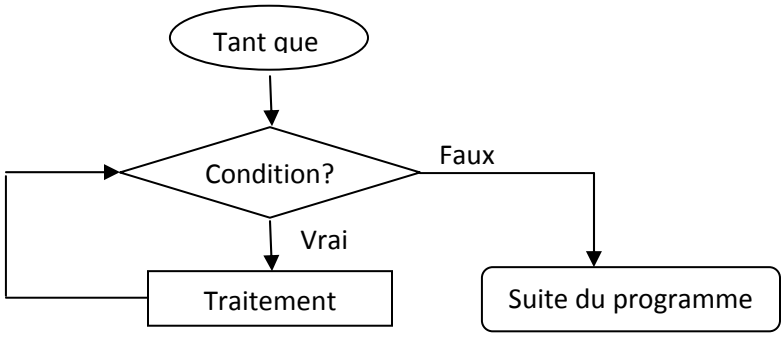
Tantque (condition) faire

        Traitement

Fintantque

- Condition : condition de maintien de la boucle.
- Traitement : Action ou ensemble d'actions à exécuter tant que la condition est vérifiée.
- Le traitement est exécuté tant que la condition est vérifiée sinon on sort de la boucle.
- Si la condition n'est pas vraie dés la première exécution, la boucle ne sera jamais exécutée (0 fois).

- Le nombre de répétition n ‘est pas connu à l’avance.
- La condition doit être initialisée avant la boucle et modifiée à l’intérieur de la boucle.



Exemple

i ← 1
tantque (i≤5)
    Ecrire (i *100)
    i← i +1
Fintantque

Exécution : i = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6
Résultat 100 200 300 400 500

Exemple

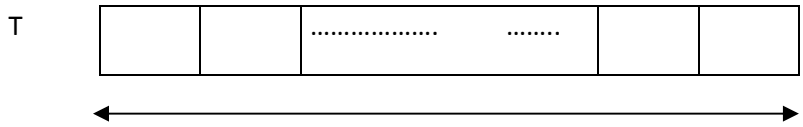
Reprendre les exercices précédents en utilisant la boucle tant que.

Chapitre 6: Les tableaux

I- Les tableaux à une dimension

1- Définition

Un tableau T est une variable structurée formée d'un nombre entier N de variables simples de même type, qui sont appelées les composantes du tableau.
Le nombre de composantes N est alors la dimension du tableau.



N composantes

On dit encore que T est ***un vecteur*** de dimension N.

2- Utilité

- Un tableau est une structure de données constituée d'un nombre fini d'éléments de **même type.**
- Lorsque plusieurs données de même type, généralement destinées au même traitement doivent être accessibles le long d'un programme, on propose d'utiliser la structure d'un tableau.

3- Composantes

Nom : identificateur d'un tableau.

Type-élément : Les éléments d'un tableau sont caractérisés par leur type (entier, réel, caractère,……..).

Indice : Tout type dont les éléments possèdent un successeur (les types scalaires), généralement de type entier.

4- Déclaration

***Nom_tab* : Tableau [premind. .deuxind] *de* type_élément**

Exemples :

T1 : Tableau [1..50] d'entier

T2 : Tableau [1..20] de réel

T3 : Tableau [1..20] de caractère

Remarque :

Il est également possible de définir un type tableau comme dans l'exemple suivant :

CONSTANTES

Nmax = 50

TYPE

Tab : Tableau [1..nmax] d'entier

VARIABLES

T : tab

5- Accès aux composantes d'un tableau

Considérons un tableau T de dimension N

- L'accès au premier élément du tableau se fait par **T[1]**
- L'accès au dernier élément du tableau se fait par **T[N]**

Exemple :

Nom : T	100	200	300	400	500
Indice :	1	2	3	4	5
Contenu	T[1]	T[2]	T[3]	T[4]	T[5]

6- Chargement d'un tableau

Ecrire un algorithme qui permet de remplir un tableau de 5 entiers.

```
ALGORITHME Chargement

VARIABLES

T : Tableau [1..5] d'entier

    i : entier

Début

    Pour i de 1 à 5 Faire

        Ecrire ("T [" , i , "] :")
```

```
            Lire(T[i])
        Fin pour
Fin
```

7- Affichage du contenu d'un tableau

```
ALGORITHME AFFICHER
VARIABLES
T : Tableau [1..5] d'entier
        i : entier
Début
        Pour i de 1 à 5 Faire
        Ecrire (T[i])
        FinPour
Fin
```

8- Méthodes de recherche dans un tableau

8-1- La recherche séquentielle

Problème : Déterminer la première position d'une valeur donnée dans un tableau de N élément. Résoudre ce problème en utilisant la notion de procédures/Fonctions

```
Algorithme RECHERCHE
CONSTANTES
        Nmax = 50
TYPE
        Tab : Tableau [1..nmax] d'entier
VARIABLES
        T : tab
        N , val : entier
/*Procédure CHARGEMEMENT*/
Procédure CHARGEMEMENT (T : tab ; N :entier)
VARIABLES
        i : entier
DEBUT
        Pour i de 1 à N Faire
                Ecrire ("T [" , i , "] :")
                Lire(T[i])
        Fin pour
FIN
/*Procédure AFFICHE*/
```

```
Procédure AFFICHE ( T : tab ; N :entier)
VARIABLES
        i : entier
DEBUT
        Pour i de 1 à N Faire
        Ecrire ( T[i])
        FinPour
FIN
/*Procédure INDICE*/
Fonction INDICE ( T : tab ; N, val :entier) :
entier VARIABLES
        i, pos : entier
DEBUT
pos← -1
i← 1
Tant que (i ≤ N et pos = -1 ) Faire
        Si (T[i] = val ) alors
                pos ←  i
        Sinon
                i ← i+1
        Finsi
FinTantque
        INDICE ← pos
FIN
/*Programme Principal*/
DEBUT ( P.P)
        Répéter
Ecrire("Donner la taille de T :")
Lire(N)
        Jusqu'à (N>1 et N<=nmax)
Ecrire (" Chargement de T ")
CHARGEMEMENT ( T , N )
Ecrire (" Affichage de T ")
AFFICHE( T , N )
Ecrire (("Donner la valeur à chercher dans T :")
        Lire(val)
Si( INDICE ( T , N, val ) = -1) alors
        Ecrire (val , "n'existe pas dans T ")
sinon
        Ecrire (val , "existe à la position", INDICE ( T , N, val ), "dans T ")
Finsi
FIN
```

8-2- La recherche dichotomique

Problème : Déterminer la première position d'une valeur donnée dans un tableau de N élément ***triés*** dans le sens croissant. Résoudre ce problème en utilisant la notion de procédures/Fonctions.

Principe :

Le principe est de décomposer le tableau T en deux sous tableaux. Trois cas peuvent se produire :

Si val = T[milieu] alors val est trouvé et la recherche est terminée.

Si val < T[milieu] alors on va chercher val dans la partie gauche du tableau T.

Si val > T[milieu] alors on va chercher val dans la partie droite du tableau T.

On poursuit la recherche tant que T[milieu] est différent de val est tant que la dimension de sous tableau reste valide.

```
Fonction Dichotomique ( T : tab ; N, val :entier) : entier
VARIABLES
        i, pos,mil, inf, sup : entier
DEBUT
pos← -1
inf← 1
sup← N
Tant que ( inf ≤ sup et pos = -1 ) Faire
        mil ← (inf + sup) div 2
        Si (T[mil] = val ) alors
                pos = mil
        Sinon
                Si (val<T[mil]  ) alors
                        sup← mil - 1
                sinon
                        inf ← mil + 1
                finsi
        Finsi
FinTantque
        INDICE ← pos
FIN
```

9- Méthodes de tri dans un tableau

9-1- Tri par sélection (par minimum)

<u>Principe</u> :

Le principe de cette méthode est simple. Elle consiste à :

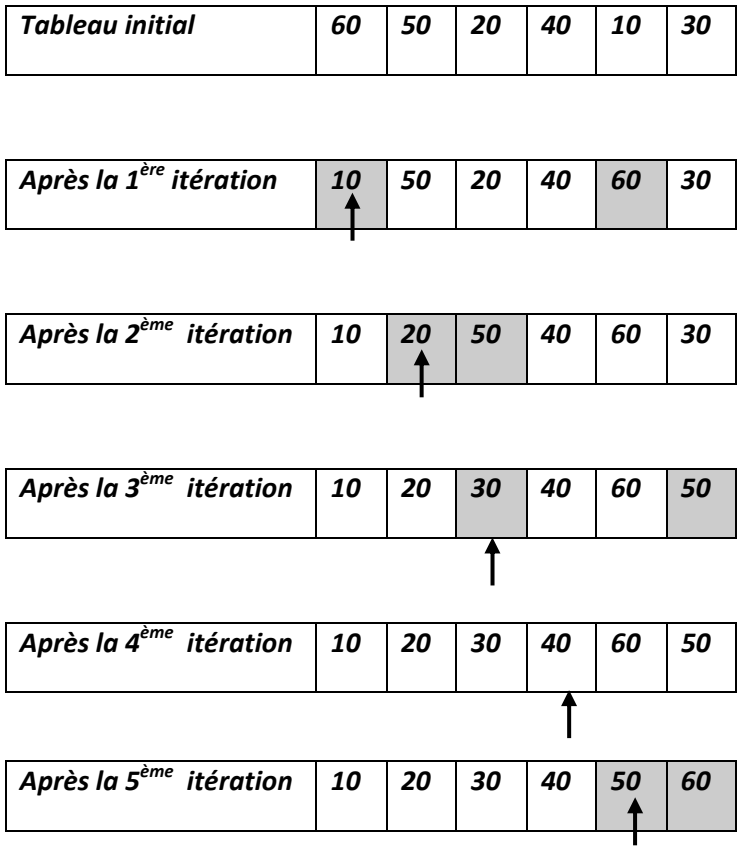
Chercher l'indice du plus petit élément du tableau T[1..n] et permuter l'élément correspondant avec l'élément d'indice 1;

Chercher l'indice du plus petit élément du tableau T[2..n] et permuter l'élément correspondant avec l'élément d'indice 2 ;

……..

Chercher l'indice du plus petit élément du tableau T[n-1..n] et permuter l'élément correspondant avec l'élément d'indice n-1;

```
Procédure TRISELECTION (T : tab ; N : entier)
VARIABLES
        i, j, aux, indmin : entier

DEBUT

Pour i de 1 à n-1 faire
                indmin  ← i
Pour j de i+1 à n faire

                Si ( T[j] < T[indmin] ) alors
                        indmin  = j
                Finsi

FinPour

 Si (i ≠ indmin ) alors
                        aux  ← T[i]
                        T[i] ← T[indmin]
                        T[indmin] ← aux
Finsi

FinPour

FIN
```



9-2- Tri à bulles

Principe **:**

Cet algorithme porte le nom de tri bulle car, petit à petit, les plus grands éléments du tableau remontent, par le jeu des permutations, enfin de tableau. Dans un aquarium il en va de même : les plus grosses bulles remontent plis rapidement à la surface que les petites qui restent collés au fonds.

Il existe plusieurs variantes de cet algorithme :

Une méthode consiste à faire descendre les plus petites valeurs au début du tableau. Dans ce cas, le tableau est parcouru de droite à gauche.

```
                                /*Procédure TRISBULLE */
Procédure TRISBULLE (T : tab ; N : entier)
VARIABLES
        i, j, aux : entier
DEBUT
        Pour i de 1 à n-1 faire
        j ← n
```

```
        Tantque (j ≠ i) faire
                Si (T[j] < T[j - 1] ) alors
                        aux  ← T[j]
                        T[j] ← T[j - 1]
                        T[j - 1] ← aux
                Finsi
                 j  ← j – 1
        FinTantque
         FinPour
FIN
/*Programme Principal*/
DEBUT ( P.P)
        N ← SAISIE_TAILLE ()
        Ecrire (" Chargement de T ")
CHARGEMEMENT ( T , N )
Ecrire (" Affichage de T  avant tri")
AFFICHE ( T , N )
        TRISBULLE (T, N)
Ecrire (" Affichage de T  après tri")
AFFICHE ( T , N )
FIN
```

Tableau initial	***50***	***30***	***20***	***40***	***10***
1ère étape	***50***	***30***	***20***	***10***	***40***
	50	***30***	***10***	***20***	***40***
	50	***10***	***30***	***20***	***40***
	10	***50***	***30***	***20***	***40***
2ème étape	***10***	***50***	***20***	***30***	***40***

10	*20*	*50*	*30*	*40*

3ème étape	*10*	*20*	*30*	*50*	*40*

4ème étape	*10*	*20*	*30*	*40*	*50*

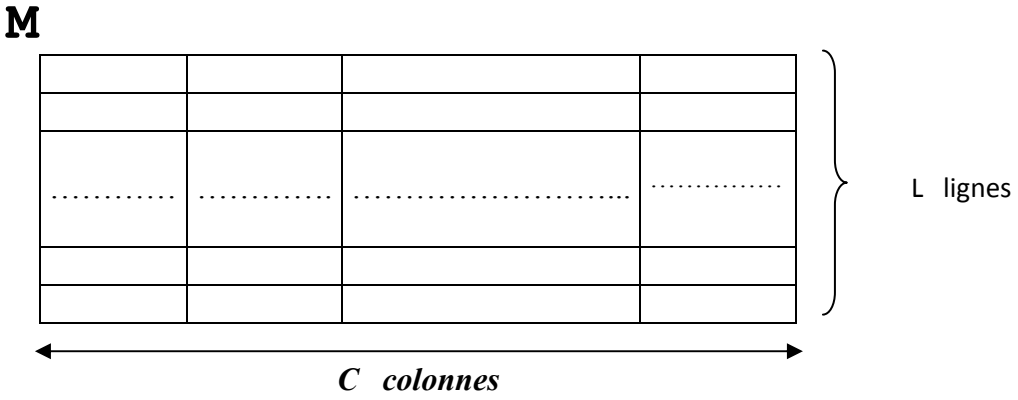
II- Les tableaux à deux dimensions

1- Définition

Un tableau à deux dimensions A et à interpréter comme un tableau (unidimensionnel) de dimension **L** dont chaque composante est un tableau (unidimensionnel) de dimension **C**.

On appelle **L** le **nombre de lignes** du tableau et **C** le **nombre de colonnes** du tableau.

Un tableau à deux dimensions contient **L*C** composantes.



2- Déclaration

***Nom_tab* : Tableau [premind. .deuxind , premind. .deuxind] *de* type_élément**

Exemples :

M1 : Tableau [1..30, 1..30] d'entier

M2 : Tableau [1..20, 1..20] de réel

M3 : Tableau [1..20, 1..20] de caractère

Remarque :

Il est également possible de définir une matrice comme dans l'exemple suivant :

CONSTANTES

NL = 30: Entier

NC = 20

TYPE

MAT : Tableau [1.. NL, 1.. NC] d'entier

VARIABLES

M : MAT

3- Accès aux composantes d'une matrice

Considérons un tableau M de L lignes et C colonnes.

- Les indices du tableau varient de 1 à L, respectivement de 1 à C.
- La composante de la N^{ième} ligne et M^{ième} colonne est notée : A[N,M].

Syntaxe :

<Nom du tableau>[<ligne> ,<colonne>]

Exemple **:** Considérons une matrice de 3 lignes et 4 colonnes

	1	***2***	***3***	***4***
1	**A[1 ,1]**	**A[1,2]**	**A[1,3]**	**A[1 ,4]**
2	**A[2 ,1]**	**A[2 ,2]**	**A[2 ,3]**	**A[2 ,4]**
3	**A[3 ,1]**	**A[3 ,2]**	**A[3 ,3]**	**A[3 ,4]**

4-Chargement d'une matrice

```
Algorithme Chargement

VARIABLES

    M : Tableau [1.. 3, 1..4] d'entier
    i ,j : entier
Début

    Pour i de 1 à 3 Faire
        Pour j de 1 à 4 Faire
            Ecrire ("M [" , i , ", " , j ,  "] :")
            Lire (M [i, j])
        Fin pour
    Fin pour
Fin
```

5-Affichage du contenu d'une matrice

```
Algorithme Afficher
VARIABLES
        M : Tableau [1.. 3, 1..4] d'entier
        i,j : entier
Début
        Pour i de 1 à 3 Faire
                Pour j de 1 à 4 Faire
                        Ecrire ( M[i, j])
                Fin pour
Fin pour
Fin
```

Exemple 1

Soient M1 et M2 deux matrices à n lignes et m colonnes. On veut écrire une procédure qui calcule les éléments de la matrice M3 = M1 + M3

Rappel:

```
/     M1     \   /      M2       \     /           M3           \
| a b c d |   | a' b' c' d' |     | a+a' b+b' c+c' d+d' |
| e f g h | + | e' f' g' h' |  =  | e+e' f+f' g+g' h+h' |
| i j k l |   | i' j' k' l' |     | i+i' j+j' k+k' l+l' |
\           /   \             /     \                     /
```

Procédure SOMME (M1 , M2, M3 : MAT ;n, m : entier)

VARIABLES

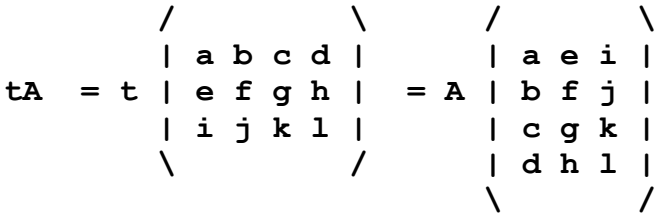
```
        i ,j : entier
Début

        Pour i de 1 à n Faire
                Pour j de 1 à m Faire
                M3[i, j] ← M1[i, j] + M2[i, j]
                Fin pour
        Fin pour
Fin
```

Exemple 2

Ecrire un algorithme qui effectue la transposition t_A d'une matrice A de dimensions N et M en une matrice de dimensions M et N.

La matrice A sera transposée par permutation des éléments. Résoudre ce problème en utilisant la notion de procédures/Fonctions.

Rappel:



Algorithme CHANGEMENT

CONSTANTES
Nmax = 50
TYPE
MAT : Tableau [1.. Nmax, 1.. Nmax] d'entier
VARIABLES
A : MAT
N , M : entier

/*Procédure SAISIE_TAILLE */

Fonction SAISIE_TAILLE() : entier

VARIABLES

nb : entier
DEBUT
Répéter
Ecrire("Donner la taille de T :")
Lire(nb)
Jusqu'à (nb>1 et nb<=nmax)
SAISIE_TAILLE ← nb

FIN

/*Procédure CHARGEMEMENT*/

Procédure CHARGEMEMENT (A : MAT ; N, M : entier)

VAR**IABLES**

i , j: entier
DEBUT
Pour i de 1 à N Faire
Pour j de 1 à M Faire
Ecrire ("A [" , i , ", " , j , "] :")

```
                    Lire (A [i, j])
                Fin pour
        Fin pour
FIN
/*Procédure AFFICHE*/

Procédure AFFICHE (A : MAT ; N, M :entier)

VARIABLES

        i, j : entier

DEBUT
        Pour i de 1 à N  Faire
                Pour j de 1 à M  Faire
                        Ecrire (A [i, j])
                Fin pour
        Fin pour
FIN

/*Procédure transposée*/

Procédure TRANSPOSEE (A : MAT ; N, M : entier)

VARIABLES

        i, j, Dmax, aux : entier

DEBUT
        Si ( N > M ) alors
        Dmax  ← N
        Sinon
        Dmax  ← M
        Finsi
        Pour i de 1 à Dmax  Faire
                Pour j de 1 à i  Faire
                        aux ← A[i, j]
                        A[i , j] ← A[j , i]
                        A[j , i] ← aux
                Fin pour
        Fin pour
FIN

/*Programme Principal*/
```

DEBUT (P.P)

Ecrire (" Saisie des tailles ")
N ← SAISIE_TAILLE ()
M ← SAISIE_TAILLE()

Ecrire (" Chargement de A ")
CHARGEMEMENT (A, N, M)
Ecrire (" Affichage de A avant transposition ")
AFFICHE (A, N, M)

TRANSPOSEE (A, N, M)

Ecrire (" Affichage de A après transposition ")
AFFICHE (A, M, N)

FIN