

Chapitre 5: Les tableaux et chaînes de caractères

1. Quelques notions: (rappel)

1.1. Identificateur :

- Un identificateur désigne le nom d'une variable, constante, type de données, procédure ou fonction...

1.2. Variable :

- Une variable possède un nom, un type, et une valeur.

1.3. Type :

- Les données peuvent être des types simples ou structurés, en plus il y a la possibilité de définir de nouveaux types de données.

- ✓ Types simples :

Exemple : entier, réel, caractère, booléen

- ✓ Types structurés :

Exemple : tableaux, chaîne de caractères, enregistrement...

2. Tableaux

- Un tableau est une structure de données regroupant un nombre fixe de variables de même type.

2.1. Vecteurs : (tableau à une dimension)

Déclaration : pour déclarer un vecteur, on utilise la syntaxe suivante :

Nom_vect [taille] : **tableau de type**

1) En précisant la taille par un nombre entier positif :

V [20] : tableau de entiers ;

2) à l'aide de constante entier positive :

CONST n ← 10;

V [n] : tableau de entiers ;

Représentation d'un Vecteur :

12.5	3.9	0.8	1.13	2.0	0.0	5.0	1.2	0.1	0.5
i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10

V [5] = 2.0

L'indice peut être :

- Une Valeur : V [5]
- Une variable : V [i]

- Une expression : $V [i*2]$

Exemple :

Écrire un algorithme qui permet de lire les moyennes de 25 étudiants, puis calcul la différence entre la moyenne de chaque étudiant avec celle de la moyenne de groupe ?

ALGORITHME Exp_Vect

```
CONST n ← 25 ;  
    VMOY [n] : tableau de réel;  
    i : entier ;  
    SMOY, MOYG : réel ;
```

Début

```
// Chargement (lecture) du tableau
```

Pour i allant de 1 à n **Faire**

```
    Ecrire (" donner la moyenne de l'étudiant N° ", i )
```

```
    Lire (VMOY [i])
```

Fin Pour

```
// Calcul de la moyenne de groupe
```

```
SMOY ← 0 ;
```

Pour i allant de 1 à n **Faire**

```
    SMOY ← SMOY+VMOY[i] ;
```

Fin pour

```
MOYG ← SMOY / N ;
```

```
Ecrire (" la moyenne du groupe est ", MOYG)
```

```
/*Calcul de la différence entre la moyenne de groupe et celle de l'étudiant*/
```

Pour i allant de 1 à n **Faire**

```
    Ecrire (" la différence de la moyenne du groupe et celle de l'étudiant ", i, " est= ",
```

```
    MOYG - VMOY[i]) ;
```

Fin pour**Fin.**

On peut écrire les deux premières boucles en une seule. Simplifier alors cet algorithme.

Remarque :

- La taille d'un tableau est fixée et ne peut être donc changée dans un programme : il en résulte deux défauts :
 - ✓ Si on limite trop la taille d'un tableau on risque le dépassement de capacité.
 - ✓ La place mémoire réservée est insuffisante pour recevoir toutes les données

2.2. Les méthodes de recherche dans un vecteur :

a) Recherche du maximum d'un vecteur :

- Elle consiste à définir le plus grand élément d'un vecteur
- Pour cela, on doit parcourir le vecteur en conservant à chaque itération le plus grand élément obtenu,
- À la fin du parcours, on obtient le maximum de tous les éléments.

Algorithme Rech_max

Const tailleM ← 100 ; // la taille maximale du tableau

Vect [tailleM] : tableaux de réel ;

Max : réel ;

i,n :entier ; // n représente le nombre réel des éléments

Début

// après la lecture de n qui représente le nombre des éléments qu'on va lire

// On suppose que les **n** éléments du vecteur ont déjà été lus.

Max ← vect[1]; //on suppose que le premier élément est le maximum

Pour i allant de **2** à **n** faire // on commence du 2^{ème} élément

Si vect[i] > max **alors**

 Max ← vect[i] ;

Finsi

Finpour;

Ecrire ('le maximum est ', max) ;

Fin.

b) Recherche séquentielle :

- L'un des premiers opérations sur les tableaux est la recherche d'un élément, son nombre d'apparition, sa ou bien ses positions.
- Pour cela, on doit parcourir tout le vecteur élément par élément et le comparer avec la valeur de l'élément à chercher.

Applications :

1. Chercher la position de la première occurrence d'un élément 5 dans un vecteur V contenant n éléments entiers ?

Algorithme recherche1

Const n←10 ;

V[n] : Tableau de entier ;

i : entier ;

Début

// On suppose que les éléments du vecteur ont déjà été lus.
// Chercher la position de la première occurrence de l'élément 5

i ← 1 ;

Tant que (i ≤ n et V[i] ≠ 5) **faire**

 i ← i + 1 ;

Fin tant que

Si (i > n) **alors**

 Ecrire ("Elément introuvable") ;

Sinon

 Ecrire ("La position de l'élément est :", i) ;

Fin.

2. Chercher le nombre d'apparition de l'élément 5 dans un vecteur V contenant n éléments, ainsi que les positions des occurrences de cet élément ?

Algorithme recherche2

Const n ← 10 ;

V[n] : Tableau de entier ;

i, nba : entier ;

Début

//Chargement du tableau

Pour i de 1 à n **Faire**

 Ecrire (" donner l'élément N° ", i) ;

 Lire (V [i]) ;

Fin pour

// Fin chargement

i ← 1 ; compt ← 0 ;

Tant que (i ≤ n) **faire**

Si (V[i]=5) **alors**

 compt ← compt + 1 ;

 Ecrire (" la position d'occurrence 5 est ", i) ;

finsi

 i ← i + 1 ;

Fin tant que

Ecrire ("le nombre d'occurrence de 5 est :", compt) ;

Fin.

c) Recherche dichotomique :

➤ Ce type de recherche s'effectue dans un tableau **ordonné** :

- 1) On divise le tableau en deux parties sensiblement égales,
- 2) On compare la valeur à chercher avec l'élément du milieu,
- 3) Si elles ne sont pas égales, on s'intéresse uniquement à la partie contenant les éléments voulus et on délaisse l'autre partie.
- 4) On recommence ces 3 étapes jusqu'à avoir un seul élément à comparer.

Application :

On suppose qu'on dispose d'un vecteur **V** de **n** éléments. On veut chercher la valeur **Val** ?

Algorithme rech_dich

```
Const n ← 100 ;  
V[n] : Tableau de entier ;  
linf, Isup, Imil , Val : entier ;  
Trouv : Booléen;
```

Début

```
linf ← 1 ; Isup ← n ;  
Trouv ← faux ;  
Tant que ((linf <= Isup) et (Trouv = faux )) Faire  
    Imil ← (linf+Isup) div 2 ;  
    Si (V[Imil] = Val) Alors  
        Trouv ← vrai ;  
    Sinon  
        Si (V [Imil] < Val) Alors  
            linf ← Imil + 1 ;  
        Sinon  
            Isup ← Imil -1 ;  
    Fin Si  
Fin Si  
Fin tant que  
Si (Trouv = vrai) Alors  
    Ecrire (Val, "existe à la position" , Imil) ;
```

Sinon

Ecrire (Val, "n'existe pas dans V");

Fin Si**Fin.****2.3. Les méthodes de tri dans un vecteur :**

- Le tri d'un tableau consiste à l'ordonner selon un sens, du plus petit au plus grand ou le sens inverse.

Exemple :

7	5	8	3	2	9
---	---	---	---	---	---

- Tri ordre croissant :

2	3	5	7	8	9
---	---	---	---	---	---

- Tri ordre décroissant :

9	8	7	5	3	2
---	---	---	---	---	---

a) Le tri par bulles : (par échange)

- Il consiste à effectuer un certain nombre de parcours du vecteur à trier.
- Un parcours consiste à aller d'un bout à l'autre du vecteur en comparant deux éléments successifs et en les permutant s'ils ne sont pas ordonnés.
- Cette comparaison remonte dans le vecteur comme une bulle en entraînant l'extremum (maximum ou minimum).

Exemple :

- 1^{er} étape

7	5	1	9	2	3
---	---	---	---	---	---

5	7	1	9	2	3
---	---	---	---	---	---

5	1	7	9	2	3
---	---	---	---	---	---

5	1	7	2	9	3
---	---	---	---	---	---

5	1	7	2	3	9
---	---	---	---	---	---

- 2^{ème} étape

5	1	7	2	3	9
---	---	---	---	---	---

1	5	7	2	3	9
---	---	---	---	---	---

1	5	7	2	3	9
---	---	---	---	---	---

1	5	2	7	3	9
---	---	---	---	---	---

1	5	2	3	7	9
---	---	---	---	---	---

➤ 3^{ième} étape

1	5	2	3	7	9
---	---	---	---	---	---

1	5	2	3	7	9
---	---	---	---	---	---

1	2	5	3	7	9
---	---	---	---	---	---

1	2	3	5	7	9
---	---	---	---	---	---

1	2	3	5	7	9
---	---	---	---	---	---

1	2	3	5	7	9
---	---	---	---	---	---

➤ 4^{ième} étape

On fait le même parcours, on trouve qu'on n'a rien à trier, donc le vecteur est trié.

Algorithme TriBulle

const n ← 6 ;

V[n] : tableau de entier ;

i : entier ;

Trier : booléen ;

Début

//on suppose que le tableau est déjà lu

Répéter

Trier ← Vrai ;

Pour i allant de 1 à n - 1 **faire**

Si (V[i] > V [i+1]) **alors**

X ← V[i];

T[i] ← V [i+1];

V [i+1] ← x;

Trier ← Faux ;

Finsi

Finpour

Jusqu'à (Trier=Vrai) ;

Fin.

b) Le tri par sélection :

- Cette méthode consiste à rechercher le minimum et à le placer en première position.
- Parcourir le reste des valeurs pour trouver le prochain plus petit élément et le placer dans la position suivante et ainsi de suite.

Remarques :

-Pour une raison de commodité, nous considérons que toutes les valeurs sont distinctes dans l'algorithme suivant.

-On peut chercher le maximum et le placer à la fin, et ainsi de suite.

Exemple

7	5	8	9	2	3
---	---	---	---	---	---

1^{er} on cherche le maximum pour les 6 éléments qui est 9, donc on va le mettre dans la dernière case du tableau

7	5	8	3	2	9
---	---	---	---	---	---

On a effectué n-1 opération pour trouver le max avec n=6.

Maintenant on ne parle que du 5 éléments

2^{ème} on cherche le max dans les (6-1) éléments restants qui est 8 et on le place dans la 5^{ème} position et on obtient le nouveau tableau

7	5	2	3	8	9
---	---	---	---	---	---

On a effectué n-2 opération pour trouver le max avec n=6.

Puis on passe au (6-2)=4 éléments du tableau

3^{ème} on refait la même chose pour les 4 éléments restants donc 7 est le maximum, on le place dans la position 4 et on obtient le tableau suivant :

3	5	2	7	8	9
---	---	---	---	---	---

On a effectué n-3 opération pour trouver le max avec n=6.

4^{ème} on itère la même chose, donc le max est 5 pour les (6-3)=3, on place 5 à la position 3.

3	2	5	7	8	9
---	---	---	---	---	---

On a effectué n-4 opération pour trouver le max avec n=6.

5^{ème} même chose que précédemment, on cherche le max dans les (6-4)=2 éléments restants et on obtient 3, on le place à la position 2.

2	3	5	7	8	9
---	---	---	---	---	---

On a effectué n-5 opération pour trouver le max avec n=6.

6^{ème} on cherche le max avec le dernier élément restant du tableau qui reste le même.

2	3	5	7	8	9
---	---	---	---	---	---

Enfin on obtient le tableau trié.

Donc le nombre d'opérations est 1+2+3+4+5 qui est une suite numérique dont la somme est :

$(n-1) n/2$.

Algorithme TriSelection

const $n \leftarrow 6$;

$V[n]$: tableau de entier ;

$i, j, \text{Min}, \text{Pos}$: entier ;

Début

//on suppose que le tableau est déjà lu

Pour i allant de 1 à n faire

$\text{Min} \leftarrow T[i]$;

$\text{Pos} \leftarrow i$;

Pour j allant de $i+1$ à n faire

Si $T[j] < \text{Min}$ alors

$\text{Min} \leftarrow T[j]$;

$\text{Pos} \leftarrow j$;

Finsi ;

Finpour ;

$T[\text{pos}] \leftarrow T[i]$;

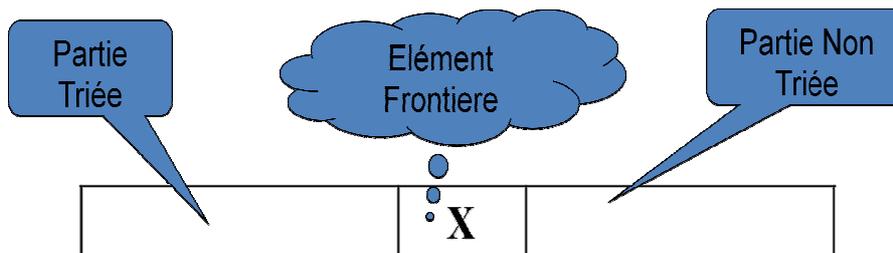
$T[i] \leftarrow \text{Min}$;

Finpour ;

Fin.

c) Le tri par insertion :

- C'est le tri que toute personne utilise naturellement quand elle a des dossiers (ou n'importe quoi d'autre) à classer.
- On prend un dossier et on le met à sa place parmi les dossiers déjà triés. Puis on recommence avec le dossier suivant.



- Tant que la partie non triée n'est pas vide,
- On prend l'élément frontière avec la partie non triée
- Et on l'insère à sa place dans la partie triée puis on avance à l'élément suivant.

pour i allant de 2 à n faire

 /* Stockage de la valeur en i */

$x \leftarrow t[i]$;

 /* Recherche du plus grand indice p */

 /* inférieur à i tel que $t[p] \leq t[i]$ */

$p \leftarrow i-1;$

Tantque ($t[p] > x$ ET $p > 0$) **faire**

$p \leftarrow p-1 ;$

FinTQ

/* Il faut insérer $t[i]$ juste après cette case */

$p \leftarrow p+1 ;$

/* Décalage des valeurs entre p et i */

Pour j allant de $i-1$ à p pas= -1 **faire**

$t[j+1] \leftarrow t[j];$

FinPour

$t[p] \leftarrow x;$

FinPour

2.4. Matrices : (tableau à deux dimensions)

Déclaration :

1)

$M [5, 10]$: tableau de réel ;

2)

CONST $n \leftarrow 5, m \leftarrow 10;$

$M [n, m]$: tableau de entiers ;

3)

n, m : réel ;

Mat $[n, m]$: tableau de entier ;

Représentation d'une Matrice:

	J=1	J=2	J=3	J=4	J=5	J=6	J=7	J=8	J=9	J=10
I=1	-4	3	14	6	67	4	2	0	7	2
I=2	1	2	3	4	5	6	7	8	9	10
I=3	9	9	3	87	76	5	2	2	2	1
I=4	1	3	2	4	-5	6	7	8	9	4
I=5	9	9	7	8	9	-7	-1	3	5	17

L'élément d'indice $[i,j]$ est celui du croisement de la ligne i avec la colonne j

$M [4,5]$ est -5

Exemple :

Soit Mat (n, m) une matrice de $n \times m$ éléments réels. Ecrire un algorithme qui permet de calculer le plus grand (max) et le plus petit (min) éléments de la matrice ?

Algorithme maxmin

Const $n \leftarrow 10$, $m \leftarrow 12$;

Mat [n, m] : tableau de réel;

max, min : réel ;

i, j : entier ;

Début

//Lecture des éléments de la matrice

Pour i allant de 1 à n **faire**

Pour j allant de 1 à m **faire**

 Lire (mat [i, j]);

 // calcule de plus grand (max) et le plus petit (min)

max \leftarrow mat [1, 1] ; min \leftarrow mat [1,1] ;

Pour i allant de 1 à n **faire**

Pour j allant de 1 à m **faire**

Si (mat [i, j] >max) **alors** max \leftarrow mat [i, j] ; **fin si**

Si (mat [i][j] <min) **alors** min \leftarrow mat [i, j] ; **fin si**

Fin Pour

Fin Pour

 Ecrire ("la plus grande valeur de la matrice", max) ;

 Ecrire ("la plus petite valeur de la matrice", min) ;

Fin.**Remarque :**

- *matrice carrée* : une matrice dont le nombre de lignes est égale au nombre de colonnes.
- Une telle matrice a une *diagonale principale* (tous les éléments pour lesquels $i=j$).
- Les éléments supérieurs à la diagonale ont leurs indices $i < j$ et Ceux inférieurs à la diagonale ont leurs indices $i > j$.

3. Chaînes de caractères :

- Une chaîne de caractères (appelée **String** en anglais) est une suite de caractères, c'est-à-dire un ensemble de symboles faisant partie du jeu de caractères, défini par le code ASCII, UTF8 etc.
- Certains langages (Pascal, Java, Basic...) disposent d'un véritable type chaîne (**String**).
- En **langage C++**, il n'existe pas de type de variable pour les chaînes de caractères comme il en existe pour les entiers (**int**) ou pour les caractères (**char**).
- Les chaînes de caractères sont en fait stockées dans un tableau de **char** dont la fin est marquée par un caractère **Nul**, de valeur **0** et représenté par le caractère **'\0'**.

Exemple :

- En mémoire la chaîne "Bonjour" est représentée ainsi :

B	O	n	j	o	u	r	\0
---	---	---	---	---	---	---	----

- Tout ce qui suit le caractère **'\0'** sera ignoré

3.1. déclarations des chaînes :

- La déclaration des chaînes de caractères se comme suit:
<Identificateur> : chaîne ;

Exemple :

S : Chaîne; // S est de type chaîne avec une taille maximale de 255 caractères.

3.2. Lecture et écriture des chaînes :

- Dans notre cours, on prendra la notation suivante pour la lecture (resp.) l'affichage des chaînes de caractères :
 - **Lecture** : Lire(S) ;
 - **Affichage** : Ecrire(S)
- On peut afficher plusieurs chaînes en adjacence en utilisant le +.

Exemple :

Algorithme Exp_chaine

S, R, T : chaîne ;

Début

S ← 'Bonjour' ;

R ← 'Mesdames' ;

T ← 'Et messieurs' ;

Ecrire(s+r+t) ; // Affichera 'Bonjour Mesdames et messieurs'

Fin.

Remarque : L'opérateur + représente la **concaténation** et non pas la somme.