Chapitre 5 : Les tableaux et les chaînes de caractères

1. Introduction

En programmation, les données sont organisées sous forme de constantes et de variables de certaines manières, pour en faciliter le traitement, et y accéder plus rapidement. Il existe différents types de données, qui peuvent être divisés en deux parties : Types simples. Tels que : entiers, réels, caractères et booléens. Types composés : les tableaux, et les structures ou enregistrements.

Supposons que nous voulions entrer les notes de 1000 étudiants, les analyser et calculer quelques statistiques. Dans ce cas, il serait déraisonnable d'utiliser 1 000 variables pour stocker des notes, puis d'écrire 1 000 instructions d'entrées (lire) dans le programme. Il est préférable d'utiliser une seule variable, qui peut contenir toutes les valeurs des notes, et d'utiliser une boucle pour les saisir. Une structure qui peut stocker plusieurs valeurs en même temps s'appelle un tableau.

Dans ce chapitre, nous aborderons les deux types de tableaux statiques, unidimensionnels et multidimensionnels. Nous verrons également que les chaînes sont un cas particulier des tables.

2. Le type tableau

2.1. Définition

Table (en anglais, array) : structure de données complexe, constituée d'un ensemble fini d'éléments homogènes (de même type), accessible par des indexes indiquant leur emplacement.

Un tableau peut être vu comme un groupe de variables, de même type, portant le même nom.

Dimension : La dimension d'un tableau est le nombre d'indices nécessaires pour identifier un seul élément.

indice: Lorsque les données sont stockées dans un tableau, l'élément est identifié par un indice qui, en C, est un entier non négatif (≥ 0). L' indice commence de 0 à N - 1 (où N est la taille du tableau).

Tableau unidimensionnel

Il est aussi appelé vecteur :on peut accéder à n'importe lequel de ses éléments par un seul indice, où chaque valeur de l'indice sélectionne un élément du tableau.

2.2. La représentation

Le tableau est représenté en mémoire par une séquence de cellules adjacentes. Une nouvelle cellule ne peut pas être supprimée ou ajoutée au tableau après sa création (statique). La figure suivante représente un tableau de 8 nombres réels.

indice	0	1	2	3	4	5	6	7
valeur	15	7	-3	0	9	2	0	-3

Il n'y a pas de différence de dessiner le tableau verticalement ou horizontalement.

2.3. La déclaration

Algorithme	C
`nomTableau [Taille] : Tableau de typeElements`	`typeElements nomTableau [Taille] ;`

Le mot « **Tableau de** » étant un mot réservé dans l'algorithme, il est utilisé pour indiquer que la variable est un Tableau.

- nomTableau: C'est le nom de l'identifiant donné au tableau (le nom de la variable).
- Taille : le nombre des éléments du tableau
- typeElements : le type des éléments du tableau, il peut être de n'importe quel type tel que entier(int), réel(float), ...

Pour déclarer plusieurs tables du même type on utilise une virgule "," en précisant la taille de chaque tableau entre deux croches [ ].

Exemple

`const N=100`	`const int N=100 ;`

`note[N] : tableau de réel` `tab1[50],tab2[20] : tableau d'entier`	`float note[N] ;` `int tab1[50],tab2[20];`

2.4. L'initialisation

En C, il est possible de spécifier des valeurs initiales pour tous les éléments du tableau dans des accolades { et } lors de la déclaration de du tableau. Les valeurs sont séparées par des virgules « , », et ces valeurs doivent être du même type.

Exemple

```
int tab[] = {15, 7, -2, 0, 9, 2, 0, -3};
```

indice	0	1	2	3	4	5	6	7
valeur	15	7	2-	0	9	2	0	3-

Remarque : Vous pouvez spécifier le nombre d'éléments entre les deux Crochets « [ ] » ou les laisser vides pour qu'ils soient calculés automatiquement.

2.5. Utilisation

Le tableau ne peut pas être traité comme un seule bloc tel que tab*10, mais chaque élément doit être traité séparément. Pour accéder à un seul élément du tableau, nous utilisons le nom du tableau avec un indice à l'intérieur de deux crochets [et], et lors du calcul de l'expression entre les deux crochets , nous obtenons une valeur du type entier. Pour accéder au premier élément du tableau tab, nous utilisons tab[0]. Pour accéder au quatrième élément, nous utilisons tab[3].

مثال `tab[5-3]←tab[tab[3]+1]` ⇔ `tab[2]←tab[0+1]` ⇔ `tab[2]←7`

le tableau devient

Indice	0	1	2	3	4	5	6	7
valeur	15	7	7	0	9	2	0	-3

Remarque : la tentative d'accès à un élément qui n'existe pas (si l'indice est supérieur ou égal à la taille du tableau ou négatif) provoque l'arrêt du programme.

2.6. La lecture d'un tableau

Pour remplir un tableau de N nombres, on utilise "lire" N fois comme :

Algorithme	C

`lire(tab[0])`
`lire(tab[1])`
…
`lire(tab[N-1])`
	`scanf("%d", &tab[0]);`
	`scanf("%d", &tab[1]);`
	…
	`scanf("%d", &tab[N-1]);`

Cependant, nous remarquons que l'instruction lire est répétée N fois, c'est-à-dire itère de 0 à N-1. Par conséquent, la boucle « **pour** » peut être utilisée. Où le compteur joue le rôle de l'indice.

Algorithme	C

pour i←0 **à** N-1 **faire**
  écrire("nb", i, "⇒")
  // Juste pour clarifier
  lire(tab[i])
finPour
	for(i = 0; i < N; i++){
	  printf("nb %d ⇒", i);
	  // Juste pour clarifier
	  scanf("%d", &tab[i]);
	}

L'instruction « **lire** » est pour remplir le tableau, et « **écrire** » pour montrer à l'utilisateur ce qui est exigé de lui.

2.7. L'affichage d'un tableau

Comme la lecture, « **écrire** » répète.

Algorithme	C

pour i←0 **à** N-1 **faire**
  écrire(tab[i])
finPour
	for(i = 0; i < N; i++){
	  printf("%d\t", tab[i]);
	}

2.8. Observations

Pour parcourir tous les éléments d'un tableau, généralement, nous utilisons la boucle « pour ».
La taille du tableau doit être spécifiée lors de la programmation (déclaration), cependant, nous pouvons donner l'impression à l'utilisateur que la taille du tableau peut être changée, en déclarant un grand tableau, puis en n'en utilisant qu'une partie. Nous demandons à l'utilisateur la taille qu'il souhaite, à condition qu'il ne dépasse pas la taille réelle du tableau.

2.9. Exemple

Écrire un programme qui reçoit les moyennes de N étudiant, où N est déterminé par l'utilisateur, puis calcule le nombre d'étudiants qui n'ont pas pris la matière. (Moyenne inférieure à 10).

Algorithme

```
Algorithme nb_ajourne
Const MAX=200
var  note[MAX] :tableau de réel
     i, aj, N :entier
début
Faire
  écrire("entrer le nbr des étudiants (<",
  MAX, ")")
  lire(N)
TQ N>MAX
pour i←0 à N-1 faire
  écrire("note ", i, "⇒")
  lire(note[i])
finPour
aj ←0
pour i←0 à N-1 faire
  si note[i]<10 alors
    aj←aj+1
  finSi
finPour
écrire("le nbr des ajournés est ", aj)
fin
```

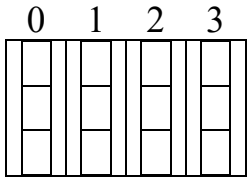
C

```
#include<stdio.h>
#define MAX 200
int main(){
float note[MAX] ;
int i, N, aj=0; // aj càd nb ajournés
// récupérer le nbr des étudiants
do{
  printf("entrer  le  nbr  des  étudiants
  (<%d)",MAX) ;
  scanf("%d",N) ;
}while (N>MAX) ;
// Remplir le tableau
for(i = 0; i < N; i++){
  printf("note %d ⇒", i);
  scanf("%d", &note[i]);
}
// calculer le nbr des ajournés
for(i = 0; i < N; i++)
  if(note[i]<10) aj++ ;
// affichage du résultat.
printf("le nbr des ajournés est %d", aj);
}
```

3. Les tableaux multidimensionnels

3.1. Définition

Un tableau bidimensionnel (également appelé matrice): Il s'agit en fait d'un tableau simple (unidimensionnel), dont les éléments, en eux-mêmes, sont des tableaux unidimensionnels. Nous voyons cela dans l'illustration ci-dessous,



Les éléments sont accessibles via deux indices, le premier spécifiant le numéro de ligne et le second spécifiant le numéro de l'élément dans cette ligne (colonne).
Ce mécanisme peut être généralisé pour créer des matrices qui ont plus de deux dimensions. Nous pouvons créer un tableau de n dimension, et l'accès à ses éléments nécessite n indices.
La disposition des indices est cruciale. L'élément M[3][2] (36) est différent de l'élément M[2][3] (28).

3.2. Représentation

La matrice est représentée ,en mémoire, par une séquence de cellules adjacentes. Une cellule ne peut être ni supprimée ni ajoutée à la matrice après sa création (statique). La figure suivante représente une matrice avec 3 lignes et 5 colonnes de nombres réels.

N ligne \ Numéro de colonne	0	1	2	3	4
0	15	7	-3	0	9
1	6	12	4	33	85
2	2	-8	17	28	-52
3	14	42	36	49	-12

3.3. Déclaration

Algorithme	C
nomMatrice [Lignes][Colonnes] : **Tableau de** typeElements	typeElements nomMatrice [Lignes][Colonnes] ;

Le mot « **Tableau de** » étant un mot réservé dans l'algorithme, il est utilisé pour indiquer que la variable est un tableau.

- nomMatrice : Le nom de l'identifiant qu'on donne à la matrice (le nom de la variable).
- Lignes : nombre de lignes
- Colonnes : le nombre de colonnes
- typeElements : Le type des éléments. Il peut s'agir de n'importe quel type. Par exemple : entier(int), réel(float), ...

Le nombre d'éléments est le nombre de lignes x le nombre de colonnes.

Exemple

```
const L=100
const C=100
M[L][C] : tableau de réel
mat1[50][30],mat2[30][20] : tableau d'entier
```

```
const int L=100 , C=200 ;
float M[L][C] ;

int mat1[50][30],mat2[30][20];
```

3.4. Initialisation

En C, les valeurs initiales de tous les éléments de la matrice peuvent être spécifiées en spécifiant les éléments de chaque ligne entre { et }, avec la déclaration de la matrice. Les valeurs sont séparées par des virgules ",", et chaque ligne est séparée par une virgule ",". Toutes les valeurs doivent être du même type.

Exemple

int mat[][] = **{{**15, 7, -3 ,0 ,9**},{**6, 12, 4,33,85**},{**2, -8, 17 ,28,-52**},{**14, 42, 36, 49, -12**}}**;

n ligne \ numéro de colonne	0	1	2	3	4
0	15	7	3-	0	9
1	6	12	4	33	85
2	2	8-	17	28	52-
3	14	42	36	49	12-

Remarque : Vous pouvez spécifier le nombre de lignes et le nombre de colonnes entre les deux crochets, ou les laisser vides pour qu'ils soient calculés automatiquement.

3.5. Utilisation

Pour accéder à un seul élément de la matrice, nous utilisons le nom de la matrice avec un indice à l'intérieur de deux crochets [et] spécifiant le numéro de ligne, et un autre indice à l'intérieur de deux crochets [et] spécifiant le numéro de colonne. Pour accéder à l'élément de la première ligne et de la première colonne de la matrice mat, nous utilisons mat[0][0].

syntaxe `mat[ligne][ colonne]`

exemple `mat[1][3]←mat[1][3]+2`

la matrice devient

N - \ Numéro de colonne	0	1	2	3	4
0	15	7	3-	0	9

1	6	12	4	35	85
2	2	8-	17	28	52-
3	14	42	36	49	12-

3.6. La lecture d'une matrice

Pour remplir la matrice M[L][C], on remplit « L » lignes. Puisque chaque ligne est un tableau unidimensionnel avec « C » éléments.

Algorithme	C

pour j←0 **à** C-1 **faire**
lire(M[0][j])
fpour
pour j←0 **à** C-1 **faire**
lire(M[1][j])
fpour
…
pour j←0 **à** C-1 **faire**
lire(M[L-1][j])
fpour
	for (j=0 ;j<C ;j++)
	scanf("%d", &M[0][j]);
	for (j=0 ;j<C ;j++)
	scanf("%d", &M[1][j]);
	…
	for (j=0 ;j<C ;j++)
	scanf("%d", &M[L-1][j]);

Cependant, nous remarquons que l'instruction pour est répétée L fois. Autrement dit, itère de 0 à L-1. Par conséquent, la boucle pour peut être utilisée.

Algorithme	C

pour i←0 **à** L-1 **faire**
pour j←0 **à** C-1 **faire**
écrire("M[", i, ",", j, "]⇒")
lire(M[i][j])
finPour
finPour
	for(i = 0; i < L; i++)
	for(j = 0; j < C; j++){
	printf("M[%d, %d] ⇒", i,j);
	scanf("%d", &M[i][j]);
	}

L'instruction « lire » pour remplir la matrice, et « écrire »est d'expliquer à l'utilisateur ce qui est exigé de lui. L'instruction "for(j..." contient deux instructions "printf" à expliquer, et "scanf" pour entrer les valeurs. "for(i..." contient une seule instruction "for(j...".

Explication de "écrire" : Supposons que i = 3 et j = 5

écrire(	"M["	i	","	j	"]⇒"
ecran	M[	3	,	5	]⇒

3.7. L'affichage d'une matrice

Comme la lecture, "écrire" répète.

Algorithme	C

pour i←0 **à** L-1 **faire**
pour j←0 **à** C-1 **faire**
écrire(M[i][j])
finPour
finPour
	for(i = 0; i < L; i++){
	for(j = 0; j < C; j++)
	printf("%d\t", M[i][j]);
	printf("\n") ;
	}

L'instruction "for(j..." contient l'instruction "printf" pour imprimer l'élément M[i][j]. for(i... contient deux instructions "for(j..." pour l'impression de la ligne i et printf("\n") pour revenir à la ligne à la fin de chaque ligne i de la matrice.

Remarque: Pour visiter tous les éléments de la matrice, nous utilisons deux boucles "pour".

3.8. Exemple

Écrivez un programme qui lit les températures par heure pour 30 jours, sous la forme d'une matrice (30 par 24), puis les affiche à l'écran, après quoi il affiche la température la plus élevée et quand elle a été enregistrée.

Algorithme	C
Algorithme nb_ajourne **Const** Jr =30 Hr=24 **var** T[Jr][Hr] :**tableau de** réel maxT :réel	#include<stdio.h> #define Jr 30 // nb lignes #define Hr 24 // nb colonnes **int** main(){

```
    i, j,maxJr,maxHr :entier
début
pour i←0 à Jr-1 faire
 pour j←0 à Hr-1 faire
  écrire("T[", i+1, ",", j, "]⇒")
  lire(T[i][j])
 finPour
finPour
pour i←0 à Jr-1 faire
 pour j←0 à Hr-1 faire
  écrire(M[i][j])
 finPour
finPour
maxT←T[0][0]
maxJr←0
maxHr←0
pour i←0 à Jr-1 faire
 pour j←0 à Hr-1 faire
  si (T[i][j]>maxT) alors
   maxT←T[i][j]
   maxJr←i
   maxHr←j
  finSi
 finPour
finPour
écrire("la température maximale est ",
   maxT," et a été enregistrée le ",
   maxJr+1, " à ", maxHr )
fin
```

```
float T[Jr][Hr] ,maxT; // max température
int i, j,maxJr,maxHr;
// Remplir les températures
for(i = 0; i < Jr; i++)
 for(j = 0; j < Hr; j++){
  printf("T[%d, %d] ⇒", i+1,j);
  scanf("%d", &T[i][j]);
 }
// afficher toutes les températures
for(i = 0; i < Jr; i++){
 for(j = 0; j < Hr; j++)
   printf("%d\t", M[i][j]);
 printf("\n") ;
}
// recherche de la température maximale
maxT=T[0][0];
maxJr=0 ;
maxHr=0 ;
for(i = 0; i < Jr; i++)
 for(j = 0; j < Hr; j++)
  if (T[i][j]>maxT){
   maxT=T[i][j];
   maxJr=i;
   maxHr=j;
  }
//affichage des résultats
printf("la température maximale est %d et
   a été enregistrée le %d à %d", maxT,
   maxJr+1, maxHr) ;
}
```

Le programme prend la matrice T des thermostats et nous renvoie la plus grande température enregistrée maxT, le jour où vous avez enregistré maxJr et l'heure où vous avez enregistré maxHr. Après avoir rempli la matrice et l'avoir affichee, nous supposons que la plus grande température est à la ligne 0 et 0 heure, puis nous visitons tous les éléments de la matrice, et si nous trouvons une température supérieure à celle stockée dans maxT, le maxT change. Ainsi, maxJr et maxHr changent. Et à la fin, nous montrons les résultats à l'écran, avec une incrementation pour maxJr parce que la ligne commence à partir de zéro 0, et le jour commence à partir de 1 un.

4. Les chaînes de caractères

4.1. Définition

La chaîne de caractères est un ensemble ordonné de caractères , 0 ou plus. Ils sont toujours entre guillemets double « " » tels que "informatique", "Bonne chance\n", "1", "3.14". En C, les tables de type char sont utilisées pour créer des chaînes. Lorsque vous lisez une chaîne de caractères à partir du clavier, chaque caractère est placé dans une zone et, lorsque les caractères sont terminés, le caractère '\0' est ajouté à la fin du texte pour indiquer sa fin. Le caractère '\0' est appelé « **null** » qui porte le code 0. Il existe une constante déclarée dans la bibliothèque stdio.h appelée **NULL** en majuscules.

```
#define NULL 0
```

Remarque: Puisque chaque caractère a un code, par exemple le code de 'A' est 65, le code de 'a' est 97, de même que le caractère '\0' a un code qui est 0.

NULL ⇔ '\0' ⇔0

4.2. Déclaration

Dans les algorithmes, nous utilisons la chaîne de caractères, tandis qu'en C, nous utilisons une table de caractères. Supposons que nous ayons la chaîne str, qui peut contenir au maximum 30 caractères, y compris '\0'. Il est déclarée comme suit :

```
var str :chaine de caractères
```

```
char str[30] ;
```

`var str[30] :caractères`	

4.3. Initialisation

Dans l'exemple suivant, nous créons un tableau de caractere et nous l'initialisant avec le mot « Welcome ».

```
char slt[] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
```

Cette instruction crée un tableau à 8 caractères (7 cases pour le mot « Welcome » et une zone contenant le caractère '\0'. Toutefois, il existe un moyen plus simple et plus rapide de créer et d'initialiser une chaîne:

```
char slt[] = "Welcome";
```

Cela conduit au même résultat, qui est la création d'un tableau à 8 caractères, se terminant par le caractère '\0'.

	0	1	2	3	4	5	6	7
slt	W	e	l	c	o	m	e	\0

La taille de la table peut également être spécifiée :

```
char slt[30] = "Welcome";
```

	0	1	2	3	4	5	6	7	8	…	28	29
slt	W	e	l	c	o	m	e	\0		…		

4.4. Affectation

Les chaînes étant des tableaux, une chaîne ne peut pas être affectée à une variable directement après sa déclaration. Et l'opération suivante est faux.

```
char slt[30];
slt = "Welcome";// erreur : affectation a un tableau.
```

Pour affecter une chaîne à une variable ou copier une variable à une autre variable, nous utilisons la fonction strcpy().

```
strcpy(slt , "Welcome" );
```

4.5. Affichage des chaînes

Le format %c peut être utilisée pour afficher la chaîne lettre par lettre, jusqu'à ce que nous atteignions le symbole '\0'.

```
i←0
TQ str[i] ≠ '\0' faire
   écrire(str[i])
   i←i+1
finTQ
```

```
for (i=0 ;str[i] != '\0' ;i++)
   printf("%c",str[i]) ;
```

La chaîne peut également être affichée directement à l'aide de la format %s.

`écrire(str)`	`printf("%s",str) ;`

4.6. Lecture des chaînes

La chaîne peut être saisie directement au format %s et sans utiliser & avant le nom de la chaîne.

`lire(str)`	`scanf("%s",str) ;`

Pour entrer du texte contenant des espaces, en C, nous utilisons l'instruction gets, définie dans la bibliothèque string.h, car scanf s'arrête au premier espace.	`#include<string.h>`
	…
	`gets(slt) ;`

4.7. Quelques fonctions spécifiques à la chaîne

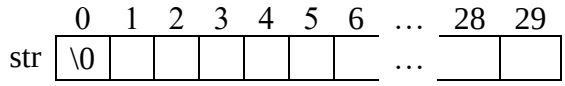
C prend en charge un large éventail de fonctions qui traitent des chaînes, qui sont définies dans la bibliothèque string.h. Parmi eux :

`strcpy(s1, s2);`	Copie la chaîne s2 dans la chaîne s1.
`strcat(s1, s2);`	pour ajouter une chaîne S2 à la fin de la chaîne S1.

`strlen(s1);`	Cette fonction renvoie la longueur de la chaîne s1.
`strcmp(s1, s2);`	Renvoie 0 si s1 et s2 sont identiques ; inférieur à 0 si s1<s2 ; supérieur à 0 si s1>s2.

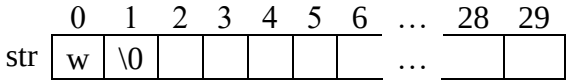
4.8. Exemples

Exemple 1 : Chaîne vide str="" ; qui est de longueur 0



Le contenu des cases n'a pas d'importance après '\0', la chaîne se termine au premier '\0'. Ainsi, n'importe quelle chaîne peut être convertie en chaîne vide en plaçant str[0]='\0' ;

Exemple 2 : chaîne contenant un seul caractère, elle est différente du type caractère. Ainsi, "w"≠'w' parce que "w" est un tableau.



Exemple 3 : écrire le programme qui entre du texte, puis convertit les lettres majuscules en minuscules et les minuscules en majuscules.

Algorithme

```
algorithme inverse
var txt :chaine[200]
    i :entier
début
 écrire(("entrer un texte")
 lire(txt)
 i←0
 TQ txt[i]≠'\0' faire
   si txt[i]>='A' et txt[i]<='Z' alors
    txt[i]=txt[i]+'a'-'A'
   sinon
    si (txt[i]>='a' et txt[i]<='z') alors
      txt[i]=txt[i]-('a'-'A');
    finSi
   finSi
 finTQ
 écrire(txt)
fin.
```

C

```
#include<stdio.h>
#include<string.h>
int main(){
 char txt[200] ;
 int i ;
 printf("entrer un texte\n") ;
 gets(txt)
 for(i=0 ;txt[i] !='\0' ;i++)
  if (txt[i]>='A'&&txt[i]<='Z')
   txt[i]+='a'-'A' ;
  else
   if (txt[i]>='a'&&txt[i]<='z')
     txt[i]-='a'-'A' ;

 printf("%s",txt) ;
 return 0 ;
}
```