Chapitre 5

Les tableaux et Les chaînes de caractères

Sommaire

Objectif

- Comprendre l'utilisation du type tableau.
- Manipuler des tableaux à une ou deux dimensions.
- Manipuler des chaînes de caractères.
- Manipuler les fonctions sur les chaines de caractères.

I. Introduction

Jusqu'à présent, nous n'avons utilisé que des variables de type prédéfini simple (entier, réel, caractère ou logique). Souvent, il est nécessaire de mémoriser une suite de valeurs numériques, de mots ou de phrases dans des variables. Mais, utiliser autant de variables n'est réellement pas rationnel. Le plus pratique est de créer des tableaux ou des chaînes de caractères, ce qui allégera les déclarations des variables et simplifiera leur lecture, leur écriture et leur accès.

Dans le présent chapitre, nous allons voir un autre type de données, c'est le type structuré "Tableau" qui nous permet de regrouper des données de même type en mémoire, ainsi que le type " chaînes de caractères".

II. Le type tableau

Exemple introductif :

Écrire un programme permettant de stocker les 200 notes du S1 des étudiants en première année MI puis les afficher avec la moyenne générale de la promotion.

Solution :

```
void main ()
{
int note = 0, somme = 0, moyenne = 0, i= 0;
for (i=0 ; i <200  ; i++)
{
printf (" donnez la note numero %d : ", i+1) ;
scanf ("%d", & note ) ;
somme +=M;
}
moyenne = somme /200 ;
printf ("La moyenne générale est : %d", moyenne );
}
```

Supposons que nous souhaitions déterminer, à partir de ces 200 notes fournies en données, combien d'étudiants ont une note supérieure à la moyenne de la classe. Pour parvenir à un tel résultat, nous devons :

- Déterminer la moyenne des 200 notes, ce qui demande de les lire toutes.
- Déterminer combien, parmi ces 200 notes, sont supérieures à la moyenne précédemment obtenue.

Vous constatez que si nous ne voulons pas être obligé de demander deux fois les notes à l'utilisateur, il nous faut les conserver en mémoire. Pour ce faire, il paraît peu raisonnable de prévoir 200 variables différentes (méthode qui, de toute manière, serait difficilement transposable à un nombre important de notes). Le type ***tableau*** va nous offrir une solution convenable à ce problème, à savoir :

- Par des déclarations appropriées, nous choisirons un nom de variable unique (par exemple *Notes*) pour repérer notre ensemble de valeurs (les notes d'un étudiant).
- Nous pourrons accéder individuellement à chacune des valeurs de ce tableau, en la repérant par un indice (ou index) précisant sa position dans le tableau.

Définitions 5.1 : Tableau

- Un tableau d'ordre n, peut être défini comme une suite de n cases mémoire consécutives (les unes à coté des autres) possédant un nom global et tel que chaque élément est caractérisé par sa position par rapport au premier arbitrairement de rang 1.

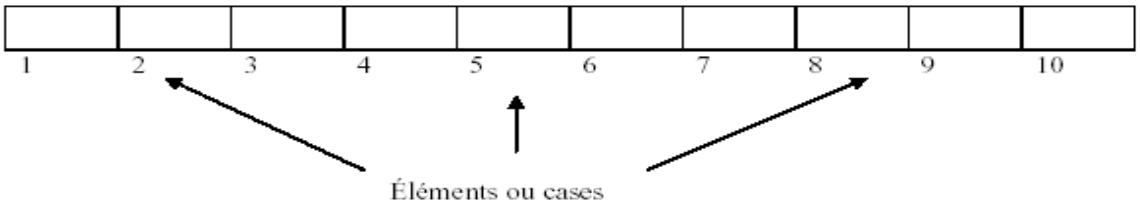
- Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle l'indice.

- Un tableau est une structure de données permettant de ranger un nombre fini d'éléments de même type. Un tableau est caractérisé par :
  - ✓ Son nom
  - ✓ Sa dimension (longueur)
  - ✓ Son type (Peut être l'un des types vus précédemment).

- Pour accéder à chaque élément du tableau, il faut écrire le nom du tableau suivi de l'indice de l'élément concerné entre crochets.

Schématiquement un tableau T de 10 éléments peut être représenté comme suit :

Tableau T



Où T : est le nom du tableau
T[1] : réfère le contenu du 1er élément du tableau T.
T[2] : réfère le contenu du 2ème élément du tableau T.
Plus généralement, si i est une variable scalaire alors T[i] réfère le contenu du ième élément du tableau T.

Pour bien utiliser un tableau on doit maitriser les tâches de déclaration, d'accès, de lecture et d'affichage.

A. Déclaration d'un tableau

La déclaration d'un tableau permet de préciser à l'ordinateur le nom du tableau ainsi que le type des éléments qui seront contenus dans le tableau. Chaque tableau à une dimension et peut être déclaré comme suit :

Syntaxe :

Algorithme	langage C
var Nom_Tableau : tableau [1.. nbr d'éléments] : <type du tableau>	Type Nom_Tableau [nbr d'éléments];

Où ***nbr d'éléments*** est le nombre des éléments dans le tableau (taille), et ***type du tableau*** indique le type des valeurs qui seront contenues dans les cases du tableau.

Un tableau doit avoir une taille fixe connue à la compilation. Cette taille peut être un nombre ou une variable constante, mais pas une variable.

Exemple :

Var
Notes : tableau [1..10] : réel ;
ou bien **// Note : est tableau de 10 éléments réels.**
Notes : tableau [10] : réel

En langage C: float Notes [10] ;

Notes	11.75	12.5	10.5	7	9.25	17	8.75	13.00	4.75	16

On déclare ici un tableau contenant 10 valeurs de type float. "[10]" représente le nombre de cases du tableau, on aura donc ici un tableau de type float de 10 cases, ces cases étant remplies de données de type **float uniquement**.

En langage C, un tableau commence à l'indice numéro 0. Notre tableau Notes de 10 float a donc les indices 0, 1, 2,... et 9. Il n'y a pas d'indice 10 dans un tableau de 10 cases. C'est une source d'erreurs très courantes pour les débutants.

B. Manipulation des composantes d'un tableau

Les éléments d'un tableau (Notes par exemple) peuvent être manipulés comme toutes autres types de variables comme suit :

```
Notes [ 1 ]  ←  x                    // x est une variable avec une valeur définie
Notes [ 2 ]  ←  3.5                  // mémorise le réel 3.5 dans la seconde case
Notes [ 3 ]  ←  Notes[1] + Notes[2]
Lire (Notes [1])                     // range la valeur saisie dans la 1ere case
Écrire (Notes [3])                   // affiche le réel contenu dans la 3eme composante
```

C. Lecture (Remplissage) d'un tableau

On peut remplir un tableau T de n éléments par une suite d'instructions de lecture comme suit : Lire (T [1]) , Lire(T[2]), … Lire (T[n])

On remarque que l'instruction Lire est répétée n fois, et afin d'éviter cette répétition, on va utiliser une des structures répétitives pour la lecture des éléments d'un tableau.

Algorithme	langage C

Var
T : Tableau [10] d'entiers ;
i : entier ;
Début
Pour i =1 **à** 10 **Faire**
Écrire ("T [", i, "] = ") ;
Lire (T[i]) ;
Fin pour
FIN
	#include<stdio.h>
	int main ()
	{
	int T[10],i;
	for (i=0;i<=9;i++)
	{
	printf("T[%d] = ",i);
	scanf("%d",&T[i]);
	}
	return o; }

D. Affichage d'un tableau

De la même façon, on peut afficher les m éléments d'un tableau T, comme suit :

Algorithme	langage C

Var
T : Tableau [10] d'entiers ;
i : entier ;
Début
Pour i =1 **à** 10 **Faire**
Écrire (T[i]) ;
Fin pour
FIN
	#include<stdio.h>
	int main ()
	{
	int T[10],i;
	for (i=0; i<=9; i++)
	{
	printf ("%d\n", T[i]); }
	return o; }

E. Initialisation d'un tableau

Maintenant que l'on sait parcourir un tableau, nous sommes capables d'initialiser toutes ses valeurs à 0 en faisant une boucle

Exemple :

```
void main ()
{
int tableau [4] , i = 0;     // Initialisation du tableau
for (i = 0 ; i < 4 ; i++)
{
tableau [i] = 0;
}
```

Il faut savoir qu'il existe une autre façon d'initialiser un tableau un peu plus automatisée en langage C. Elle consiste à placer les valeurs une à une entre accolades, séparées par des virgules.

Exemple :

```
void main ()
{
int tableau [4] = {0, 0, 0, 0} ;
}
```

On peut également définir les valeurs des premières cases du tableau, toutes celles que vous n'aurez pas renseignées seront automatiquement mises à 0.

Exemple :

```
void main ()
{
int tab1 [4] = {0, 0, 0, 0};          // 0, 0, 0, 0
int tab2 [6] = {10 , 23};             // 10, 23, 0, 0, 0, 0
int tab3 [4] = {0};                   // 0, 0, 0, 0
int tab4 [5] = {1};                   // 1, 0, 0, 0, 0, 0,
}
```

Voici la solution complète résolvant le problème posé dans l'exemple introductif :

Algorithme	langage C
Algorithme exemple_intro ; **Var** Notes : tableau [1..200] : réel ; Somme, moyenne : réel ; i , N : entier ;	# include <stdio .h> main () { int T [200] , i , N ; double Somme, moyenne ;

Début
Pour i = 1 à 200 faire
Ecrire (‘donner la note de l’élève N° ‘, i) ;
Lire (Notes[i]) ;
Fin pour
/* calcul de la moyenne */
Somme ← 0 ;
Pour i = 1 à 200 faire
Somme ← Somme + Notes[i] ;
Fin pour
moyenne ← Somme/200 ;
N ← 0 ;
Pour i = 1 à 200 faire
Si (Notes[i] > moyenne) **alors**
N ← N +1 ;
Fsi
Fin pour
Ecrire (‘moyenne de ces 200 notes = ‘ , moyenne);
Ecrire (N , ‘ élèves ont plus de cette moyenne ‘);
Fin
	for (i=0 ; i <200 ; i++)
	{
	printf ("donnez la note de l’élève N° %d : ", i+1) ;
	scanf ("%d", &t[i]) ;
	}
	Somme = 0 ;
	for (i=0 ; i <200 ; i++) Somme += T[i] ;
	moyenne = Somme/200 ;
	N = 0 ;
	for (i=0 ; i <200 ; i++)
	if (T[i] > moyenne) N ++ ;
	
	printf ("moyenne de ces 200 notes = %f\n", moyenne) ;
	
	printf ("%d élèves ont plus de cette moyenne ", N) ;
	}

III. Les tableaux multidimensionnels

Dans cette section on ne garde que les tableaux à deux dimensions (matrices) vu leur utilité pratique, notamment en mathématiques.

Exemple introductif :

Considérons un tableau NOTES_ ASD1 à une dimension pour mémoriser les notes de 200 étudiants du module ASD1:

Var

NOTES_ ASD1 : tableau [200] : réel ;

14	13	9.50	12	11	10.50	08	…..	…..	…..	…..	……	16

Pour mémoriser les notes des étudiants dans les 08 matières d'un trimestre, au lieu d’utiliser 08 tableaux à une dimension de taille 200 (ce qui est énorme), nous pouvons rassembler plusieurs de ces tableaux uni-dimensionnels dans un seul tableau NOTES à deux dimensions

Var

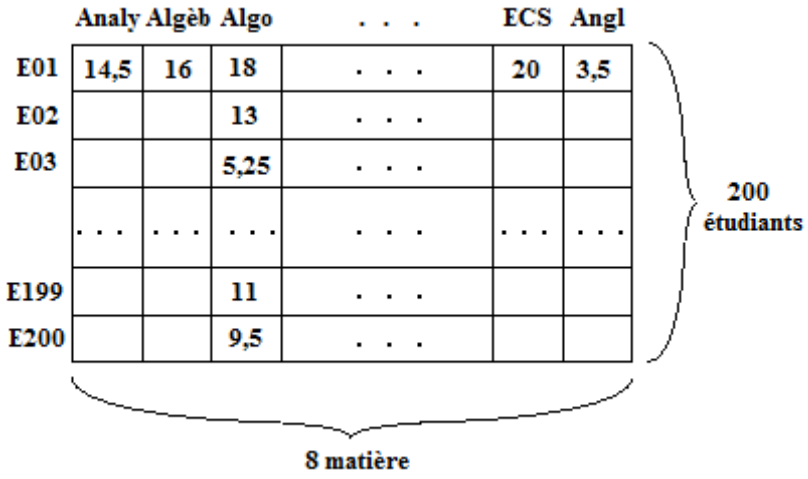
NOTES : tableau [1..8,1.. 200] : réel ;

- Dans une ligne nous retrouvons les notes de tous les étudiants dans une matière.
- Dans une colonne, nous retrouvons toutes les notes d'un étudiant.

Définitions 5.2 : Tableaux à deux dimensions

Un tableau à deux dimensions est à interpréter comme un tableau (uni-dimensionnel) de dimension N dont chaque composante est un tableau (uni-dimensionnel) de dimension M.

On appelle N le ***nombre de lignes*** du tableau et M le ***nombre de colonnes*** du tableau. N et M sont alors les deux ***dimensions*** du tableau. Un tableau à deux dimensions contient donc ***N*M composantes***.



On dit qu'un tableau à deux dimensions est ***carré***, si N est égal à M.

A. Déclaration d'un tableau à deux dimensions

Comme pour les tableaux à une dimension, une matrice est caractérisée par:

- Le nom
- La taille de la matrice (nombre de ligne x nombre de colonnes)
- Le type des éléments de la matrice.

Chaque élément dans une matrice est caractérisé par le numéro de la ligne et le numéro de la colonne

Syntaxe :

Algorithme	langage C
Var Nom_Tabl : tableau[1..<DimLigne>,1..<DimCol>]: type	Type Nom_Tabl [DimLigne] [DimCol] ;

Exemple:

Mat : tableau [8, 200] des entiers ;
int Mat [8] [200] ;

B. Accès aux éléments d'un tableau à deux dimensions

On accède à chaque élément du tableau par deux indices, un indice pour préciser le numéro de ligne et le second pour préciser le numéro de colonne.

Exemple : Considérons un tableau M MAT de dimensions 5 et 4.

MAT

5	6	3	22
0	0	1	8
4	2	8	-1
7	50	78	0
19	27	5	15

MAT [3,2] = 2 : l'élément de la 3ème ligne et la 2ème colonne

Remarque : le premier indice représente toujours la ligne et le second représente toujours la colonne

En langage C :

- *Les indices du tableau varient de 0 à N-1, respectivement de 0 à M-1.*
- *L'élément de la ième ligne et jème colonne est noté : A[i-1][j-1]*

C. Lecture d'un tableau à deux dimensions

On utilise deux boucles « Pour » imbriquées. L'indice i est utilisé pour parcourir les lignes et l'indice j pour parcourir les colonnes.

Algorithme	langage C

Var
MAT : Tableau [n, m] d'entiers ;
i, j: : entier ;
Début
Pour i =1 **à** n **Faire**
Pour j =1 **à** m **Faire**
lire (MAT[i, j]) ;
Fin pour
Fin pour
FIN
	#include<stdio.h>
	int main()
	{
	int MAT[3][2] ;
	int i, j;
	for (i = 0 ; i < 3; i++)
	{
	for (j = 0; j < 2; j++)
	{
	printf("MAT [%d,%d]= ",i,j);
	scanf("%d",&MAT[i][j]);
	}
	}
	return o;
	}

On peut remplir la matrice de deux manières différentes ou bien par lignes ou par colonnes. Pour lire colonne par colonne, on inverse juste i et j

D. Affichage d'un tableau à deux dimensions

De la même façon, on peut afficher les éléments de la matrice MAT comme suit :

Algorithme	langage C

Var
MAT : Tableau [n, m] d'entiers ;
i, j: : entier ;
Début
Pour i =1 **à** n **Faire**
Pour j =1 **à** m **Faire**
Ecrire (MAT [i, j]) ;
Fin pour
Fin pour
FIN
	#include<stdio.h>
	int main()
	{
	int MAT[3][2] ;
	int i, j;
	for (i = 0 ; i < 3; i++)
	{
	for (j = 0; j < 2; j++)
	{
	printf("tab[%d,%d]= %d \n", i, j, tab[i][j]);
	}
	}
	return o;
	}

E. Initialisation d'un tableau à deux dimensions

Lors de la déclaration d'un tableau, on peut initialiser les éléments du tableau, en indiquant la liste des valeurs respectives entre accolades. À l'intérieur de la liste, les éléments de chaque ligne du tableau sont encore une fois comprises entre accolades.

Pour améliorer la lisibilité des programmes, on peut indiquer les composantes dans plusieurs lignes.

Exemple :

```
int A[3][10] ={{ 0,10,20,30,40,50,60,70,80,90},
               {10,11,12,13,14,15,16,17,18,19},
               { 1,12,23,34,45,56,67,78,89,90}};
```

- Lors de l'initialisation, les valeurs sont affectées ligne par ligne en passant de gauche à droite.
- Nous ne devons pas nécessairement indiquer toutes les valeurs : Les valeurs manquantes seront initialisées par zéro.

Revenons à l'exemple introductif, le tableau notes contient les notes de 200 étudiants dans 8 matières, l'algorithme suivant permet de calculer la moyenne générale de chaque élève.

```
Algorithme moy_gen ;
Var
      notes : tableau[1..8,1..200] : réel ;
      i, j : entier ;
      Som , Moy : réel ;

 Début
     Pour j = 1 à 200 faire
         Ecrire ( ' donner les 08 notes de l'étudiant N° ', j )
         Pour i = 1 à  8  faire
            Ecrire (' note N° ', i ) ;
            Lire ( notes [i , j]) ;
         Fin pour
     Fin pour
     Pour j = 1 à 200 faire
         Som ← 0 ;
         Pour i = 1 à  8  faire
            Som ← Som +notes[i , j]) ;
         Fin pour
         Moy ← Som/10 ;
         Ecrire ('l'étudiant N° ', j ,' a pour moyenne ',Moy )
     Fin pour
 Fin .
```

IV. Les chaînes de caractères

Les chaînes de caractères sont en fait des tableaux de caractères. Leur manipulation est donc analogue à celle d'un tableau à une dimension, mais elles sont enrichies par d'autres fonctions spéciales facilitant leur manipulation.

A. Les caractères

Comme nous avons noté précédemment, ce type représente le domaine des caractères qui contient les lettres alphabétiques, les caractères numériques, les caractères spéciaux (., ?, !, <, >, =, *, +, ...etc), et le caractère espace. Si une variable est déclarée de type caractère, il va occuper un octet en mémoire.

Comme la mémoire ne peut stocker que des nombres, on a inventé une table qui fait la conversion entre les nombres et les lettres (la table ASCII En C,). Ainsi le nombre 65 par exemple équivaut à la lettre A. il suffit d'écrire cette lettre entre apostrophes, comme ceci : 'A' et 'A' sera donc remplacé par la valeur correspondante : 65.

Exemple :

int main ()
{
char lettre = 'A';
printf ("%c\n", lettre);
return 0;
}
	int main ()
	{
	char lettre = 'A';
	printf ("%d\n", lettre);
	return 0;
	}

Affichage à l'écran : A	***Affichage à l'écran :*** 65

Qu'est-ce que le code ASCII ?

Le code ASCII (*American Standard Code for Information Interchange*) est un code standardisé qui permet d'unifier la représentation des caractères ainsi que la communication entre les ordinateurs.

Le code ASCII, ou table ASCII, est basé sur un principe assez simple où chaque caractère (chiffre, lettre, etc.) possède un code numérique pour pouvoir être stocké et interprété par un ordinateur.

Le code ASCII

American Standard Code for Information Interchange

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	á	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	é	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	≡
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ò	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

Figure 5.1 : La table ASCII

Dans sa première version, le code ASCII représente les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Ensuite, il a été étendu pour utiliser 8 bits (256 caractères, de 0 à 255) afin de permettre le codage des caractères nationaux (non seulement anglais tels que les caractères accentués comme : ù, à, è, é, â,...etc) et les caractères semi-graphiques.

B. Déclaration des chaînes de caractères.

Syntaxe :

Algorithme	langage C
Var Nom_variable : chaîne de caractère [dim]; *Ou bien* Nom_variable : chaîne de caractère ;	char Nom_variable [dim];

Exemple :

texte : chaîne de caractère [10];
char texte[10];

En langage C, le compilateur réserve (dim-1) places en mémoire pour la chaine de caractères. Une chaîne de caractère doit impérativement contenir un caractère spécial à la fin de la chaîne, appelé « *caractère de fin de chaîne* ». Ce caractère s'écrit '\0'.

- Le caractère '\0' permet tout simplement d'indiquer la fin de la chaîne.
- Par conséquent, pour stocker le mot " Salut " (qui comprend 5 lettres) en mémoire, il ne faut pas un tableau de 5 char, mais de 6 .

Exemple :

char texte [6] = " Salut ";

'S'	'a'	'l'	'u'	't'	'\0'

- En tapant entre guillemets la chaîne que vous voulez mettre dans votre tableau, le compilateur C calcule automatiquement la taille nécessaire. C'est-à-dire qu'il compte les lettres et ajoute 1 pour placer le caractère '\0'.
- Il écrit ensuite une à une les lettres du mot _ Salut _ en mémoire et ajoute le '\0'.

Exemple :

```
int main ()
{
char chaine [] = " Salut ";
printf ("%s", chaine );
return 0;
}
```

Il y a toutefois un défaut : ça ne marche que pour l'initialisation ! Vous ne pouvez pas écrire plus loin dans le code : chaine = "Salut".

C. Lire une chaîne de caractères

On peut lire une chaîne de caractères entrée en utilisant la fonction ***scanf*** et le format %s. Une chaîne étant un pointeur, on n'écrit pas le symbole &.

Exemple :

scanf ("%s",texte);

Remarque: *scanf* ne permet pas la saisie d'une chaîne de caractères comportant des espaces. Les caractères saisis à partir de l'espace ne sont pas pris en compte.

On peut aussi lire une chaîne de caractères en utilisant la fonction ***gets*** non formatée (n'utilise pas le format %s) .

Exemple :

gets (texte);

D. Afficher une chaîne de caractères

On peut utiliser la fonction ***printf*** et le format %s, comme on peut aussi utiliser la fonction ***puts*** non formatée.

Exemple :

puts (texte); *est équivalent à* **printf**("%s", texte);

E. Fonctions sur les chaines de caractères

La bibliothèque <***string.h***> fournit une multitude de fonctions pratiques pour le traitement et la manipulation des chaînes de caractères. Voici quelques fonctions les plus fréquemment utilisées:

- **strlen : calculer la longueur d'une chaîne**

strlen(s) est une fonction qui calcule la longueur de la chaîne de caractère « ***s*** » sans compter le caractère '\0' final.

Exemple :

```
int main ()
{
char chaine [] = " Salut ";
int longChaine = 0;
longChaine = strlen ( chaine );
printf ("La chaine %s fait %d caractères de long ", chaine , longChaine );
return o;
}
```

- **strcpy : copier une chaîne dans une autre**

La fonction ***strcpy(s,t)*** permet de copier une chaîne « ***t*** » à l'intérieur de la chaîne « ***s*** ».

Exemple :

```
int main ()
{
char chaine1 [] = " Texte ", chaine2 [50];
strcpy (chaine2 , chaine1 ); // On copie " chaine1 " dans " chaine2 "
printf ("chaine1 vaut : %s\n", chaine1 );
printf ("chaine2 vaut : %s\n", chaine2 );
return 0;
}
```

La fonction ***strncpy(s,t,n)*** copie au plus « n » caractères d'une chaîne « ***t*** » vers une autre chaîne « ***s*** ».

- **strcat : concaténer 2 chaînes**

La fonction ***strcat(s,t)*** ajoute une chaîne « ***t*** » à la fin d'une autre chaîne « ***s*** ». On appelle cela la concaténation.

Exemple :

```
int main ()
{
char chaine1 [100] = " etudiants ", chaine2 [] = " MI ";
strcat ( chaine1 , chaine2 );
printf (" chaine1 vaut : %s\n", chaine1 );
printf (" chaine2 vaut toujours : %s\n", chaine2 );
return o;
}
```

La fonction ***strcat(s,t,n)*** ajoute au plus « n » caractères d'une chaîne « ***t*** » à la fin d'une autre chaîne « ***s*** ».

- **strcmp : comparer 2 chaînes**

La fonction ***strcmp(s,t)*** compare la chaîne « ***s*** » et la chaîne « ***t*** » lexicographiquement et fournit un résultat :

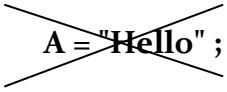
zéro si « s » et « t » sont identiques.
négatif si « ***s*** » précède « ***t*** ».
positif si « ***s*** » suit « ***t*** ».

Exemple :

```
int main () {
char chaine1 [] = " Texte de test ", chaine2 [] = " Texte de test ";
if ( strcmp ( chaine1 , chaine2 ) == 0)
      printf ("Les chaines sont identiques \n");
else
   printf ("Les chaines sont differentes \n");
return o;    }
```

Remarques :

- Comme le nom d'une chaine de caractères représente une adresse fixe en mémoire, on ne peut pas 'affecter' une autre chaine au nom d'un tableau :



  Il faut bien copier la chaine, caractère par caractère, ou utiliser la fonction **strcpy** : **strcpy(A, "Hello");**

- La concaténation de chaines de caractères en C ne se fait pas par le symbole '+' comme en langage algorithmique. Il faut ou bien copier la deuxième chaine caractère par caractère ou bien utiliser la fonction **strcat**.

- La fonction **strcmp** est dépendante du code de caractères et peut fournir différents résultats sur différentes machines.

V. Conclusion

Nous avons vu à travers ce chapitre comment utiliser des tableaux et des chaînes de caractères. Dans la pratique, les types de tableaux les plus utilisés sont les vecteurs (tableaux à une seule dimension) et les matrices (tableaux à deux dimensions).