Les tableaux et les chaînes de caractères

Vous devez avoir remarqué que lorsque vous avez besoin d'un certain nombre de variables du même type, il n'est pas pratique de déclarer beaucoup de variables et la variable à utiliser ne peux pas changer au cours de l'exécution du programme.

Les tableaux sont la pour ça. Logique, c'est le titre de ce tutorial ;-). Ils servent aussi à gérer les chaînes de caractères.

Les tableaux simples

La déclaration d'un tableau se fait comme suit :

```
type_de_donnee nom_du_tableau [nombre_d_elements] ;
```

- *nombre_d_elements* ne peux pas être une variable « standard », il doit être une valeur immédiate ou une variable déclarée avec le mot clé **const**.
- *nom_du_tableau* contient *nombre_d_elements* éléments, leurs indices vont de 0 à *nombre_d_elements* - 1.
- Pour accéder au différentes valeurs du tableau, on utilise soit des valeurs immédiates, soit des variables.

Voici un petit exemple :

```
int n[10] ; // on déclare un tableau de 10 entiers indexés de 0 à 9
int i ;

for(i = 0 ; i <= 9 ; i++)
  n[i] = i ; /* chaque élément tu tableau a pour valeur son indice :
                                                                    -  n[0] = 0
                                                                    -  n[1] = 1
                                                                    -  n[2] = 2
                                                                    -  …
                                                                    -  n[9] = 9 */

for(i = 0 ; i <= 9 ; i++)
  cout << "L'element " << i << " du tableau a pour valeur : " << n[i] << "\n" ;

// ci dessus, on affiche l'index et la valeur de chacun des 10 éléments du tableau.
// A chaque fois, on accède au ième élément de n par n[i]
```

Nous avons vu ci-dessus un tableau contenant des **int** mais un tableau peux contenir n'importe quel type de valeur : **bool, short, unsigned int, long int, char**…

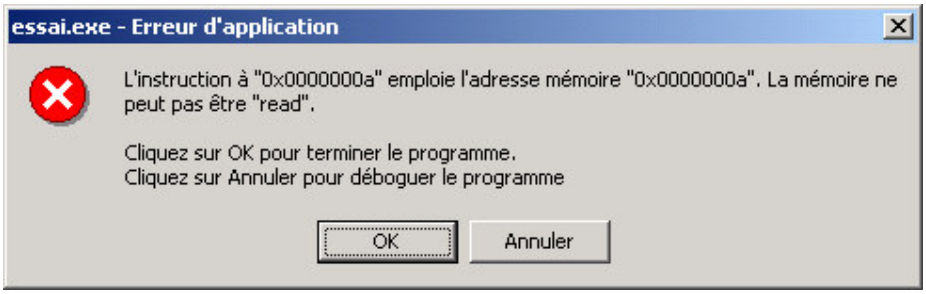
Il incombe au programmeur de s'arranger pour ne pas lire ou écrire en dehors des limites de son tableau. C++ n'effectue aucune vérification.

Voici un exemple de ce qu'il ne faut pas faire :

```
int n[10] ;

for(int i = 0 ; i < 11 ; i++)
  n[i] = i;
  // quand I vaudra 10, l'instruction n[10] = 10 posera un problème
```

Une erreur comme vue ci-dessus est fréquente. Voici le résultat :



Avouez que ça fait tout de suite pas très sérieux dans un programme. C'est pourquoi il faut TOUJOURS faire très attention, lorsqu'on utilise des tableaux, de ne pas écrire en dehors de la plage permise.

Tableaux a plusieurs dimensions

Nous avons vu ici un tableau à une seule dimension mais les tableaux peuvent être multidimensionnels. Les tableaux à plus de deux dimensions sont très lourds à gérer et peuvent vite occuper beaucoup d'espace en mémoire. C'est pourquoi je ne vous parlerai que de tableaux à deux dimensions.

La déclaration d'un tableau d'**int** a deux dimensions se fait comme ci-dessous :

```
type_de_donnee nom_du_tableau [nombre_de_lignes][nombre_de_colonnes] ;
```

Soit un tableau d'**int** déclaré comme suit :

```
int nombres[3][5] ; // déclaration d'un tableau d'int de 3 lignes et de 5 colonnes
```

Voici comment on pourrait le représenter :

	j = 0	j = 1	j = 2	j = 3	j = 4
i = 0	nombres[i][j]	nombres[i][j]	nombres[i][j]	nombres[i][j]	nombres[i][j]
i = 1	nombres[i][j]	nombres[i][j]	nombres[i][j]	nombres[i][j]	nombres[i][j]
i = 2	nombres[i][j]	nombres[i][j]	nombres[i][j]	nombres[i][j]	nombres[i][j]

$0 \leq i \leq$ nombre_de_lignes – 1
$0 \leq j \leq$ nombre_de_colonnes – 1

Remarque
L'utilisation des tableaux à deux dimensions est, dans bien des cas, pratique sinon indispensable. Néanmoins, il faut garder une chose en tête : La mémoire utilisée par un tableau à deux dimensions (ou généralement multidimensionnels) est importante.

Elle se calcule par le produit du nombre d'éléments de chaque dimension le tout multiplié par la taille du type de valeur (en octet).

Imaginons que nous utilisons un tableau d'**int** à deux dimensions pour créer une table de multiplication allant jusqu'à 100 * 100.

La mémoire occupée par un tel tableau (en comptant que la taille de l'**int** est de 4 octets) est :
100 * 100 * 4 = 40000 octets

Dans certain cas, il faudra chercher un moyen de réduire un tableau ou son nombre de dimension afin de ne pas gaspiller de la mémoire.

Exemple
Nous allons faire un petit exemple : Nous allons créer un tableau de 10 * 10 représentant une table de multiplication :

```
int table[10][10] ; // 10 lignes (0 à 9) et 10 colonnes (0 à 9)
int i, j ;

// creation de la table
for(i = 0 ; i < 10 ; i++) // ici, i varie de 0 à 9, on parcourt les lignes
  for(j = 0 ; j < 10 ; j++) // idem mais pour les colonnes
    table[i][j] = (i + 1) * (j + 1) ; // + 1 dans les deux cas car notre table par de 1
                                      // et non de 0 et va jusqu'à 10 et non 9

// affichage de la table
for(i = 0; i < 10; i++) // parcourt des lignes
{
  for(j = 0; j < 10; j++) // parcourt des colonnes
  {
    // si le nombre est plus petit que 10 (si il contient qu'un seul chiffre) on
    // affiche un 0 (uniquement pour des raisons de mise en page)
    if(table[i][j] < 10)
      cout << "0";

    // on affiche le nombre et une séparation
    cout << table[i][j] << " |";
  }

  // a chaque fin de ligne, on va à la ligne
  cout << "\n";
}
```

Chaînes de caractères

Nous n'avons vu que des tableaux contenant des **int**. Mais il est possible de faire des tableaux contenant des **bool**, des **short** et de tout autre type de données (personnalisé ou non).

Il est donc possible de créer et gérer un tableau de **char** ou **tableau de caractères**. C'est ce moyen que nous allons utiliser pour gérer les chaînes de caractères. Remarquez qu'il est possible d'utiliser le type de données **string** que nous n'allons pas étudier.

Lorsque nous allons utiliser un tableau de **char**, chaque caractère occupera une place dans le tableau, que ce soit une lettre, un chiffre, un caractère spécial ou un espace.

Chaque chaîne de caractère doit être terminée par un zéro de terminaison, il se note '\0', il est aussi appelé **zéro binaire**.

Chaque caractère est représenté par un code nommé **code ASCII**. Chaque caractère a le sien allant de 0 à 255. Certains ne sont pas utilisés fréquemment car ils ne représentent pas des caractères affichable, comme le 10 qui est le '\n' ou retour chariot.

Voici quelques codes ASCII :

!	33	C	67	e	101	‡	135	©	169
"	34	D	68	f	102	ˆ	136	ª	170
#	35	E	69	g	103	‰	137	«	171
$	36	F	70	h	104	Š	138	¬	172
%	37	G	71	i	105	‹	139	-	173
&	38	H	72	j	106	Œ	140	®	174
'	39	I	73	k	107	□	141	¯	175
(	40	J	74	l	108	Ž	142	°	176
)	41	K	75	m	109	□	143	±	177
*	42	L	76	n	110	□	144	²	178
+	43	M	77	o	111	‘	145	³	179
,	44	N	78	p	112	’	146	´	180
-	45	O	79	q	113	“	147	µ	181
.	46	P	80	r	114	”	148	¶	182
/	47	Q	81	s	115	•	149	·	183
0	48	R	82	t	116	–	150	¸	184
1	49	S	83	u	117	—	151	¹	185
2	50	T	84	v	118	˜	152	º	186
3	51	U	85	w	119	™	153	»	187
4	52	V	86	x	120	š	154	¼	188
5	53	W	87	y	121	›	155	½	189
6	54	X	88	z	122	œ	156	¾	190
7	55	Y	89	{	123	□	157	¿	191
8	56	Z	90	|	124	ž	158	À	192
9	57	[	91	}	125	Ÿ	159	Á	193
:	58	\	92	~	126		160	Â	194
;	59	]	93	□	127	¡	161	Ã	195
<	60	^	94	€	128	¢	162	Ä	196
=	61	_	95	□	129	£	163	Å	197
>	62	`	96	‚	130	¤	164	Æ	198
?	63	a	97	*f*	131	¥	165	Ç	199
@	64	b	98	„	132	¦	166	È	200
A	65	c	99	…	133	§	167	É	201
B	66	d	100	†	134	¨	168	Ê	202

Ë	203	Ö	214	á	225	ì	236	÷	247
Ì	204	×	215	â	226	í	237	ø	248
Í	205	Ø	216	ã	227	î	238	ù	249
Î	206	Ù	217	ä	228	ï	239	ú	250
Ï	207	Ú	218	å	229	ð	240	û	251
Ð	208	Û	219	æ	230	ñ	241	ü	252
Ñ	209	Ü	220	ç	231	ò	242	ý	253
Ò	210	Ý	221	è	232	ó	243	þ	254
Ó	211	Þ	222	é	233	ô	244	ÿ	255
Ô	212	ß	223	ê	234	õ	245		
Õ	213	à	224	ë	235	ö	246		

Nous allons donc créer un petit programme qui demande à l'utilisateur d'entrer son prénom et affiche un message de bienvenue. La longueur maximale du prénom qu'il sera possible d'entrer sera 20 caractères. Si un nombre supérieur de caractère est entré, on aura droit au beau message d'erreur vu ci-dessus (si vous êtes sous windows).

Nous allons récupérer le prénom avec cin, défini dans **iostream**.

```
#include <conio.h>

#include <iostream>

using namespace std;

int main()
{
  // 20 caractères + 0 final
  char prenom[21];

  cout << "Bonjour, entrez votre prenom:\n";
  cin >> prenom;

  cout << "\nSalut, " << prenom << "\n\n";

  cout << "Appuyez sur une touche pour terminer le programme...";
  getch();
  return 0;
}
```

Il faut remarquer que cin ne prend pas les espaces en compte. Si vous rentrez Toto Durand, seul *Salut, Toto* sera affiché.

Si vous possédez Visual Studio (ou tout autre compilateur permettant d'exécuter le programme en mode pas à pas et de regarder le contenu des variables), vous pourrez voir que le caractère situé après le dernier utilisé (prenom[4] si on entre Toto, par exemple) est un 0 dont le code ASCII est….. 0

Si vous voulez prendre en compte les espaces, il faut utiliser **cin.getline()**. Le premier paramètre est le tableau qui recevra la chaîne lue et le deuxième est la taille de ce tableau qui peux être obtenue par **sizeof**.

Petite explication sur sizeof

sizeof(type) renvoie la taille, en octets, de **type**. **type** peux être un type de base (**int**, **char**, **double**, …) ou un type personnalisé (classe, structure, …).

Si on considère un tableau déclaré comme suit :

```
int tableau[10];
int taille = sizeof(tableau);
```

La variable **taille** vaudra donc 40 (10 * 4 octets). Si on divise ce nombre par la taille d'**int**, on aura le nombre d'éléments du tableau :

```
int tableau[10];
int taille = sizeof(tableau) / sizeof(int); // taille = 40 / 4 = 10
```

Ici, la variable **taille** vaudra 10.

Revenons à notre **cin.getline()**.

```
char nom_prenom[64];
cout << "Entrez votre nom et votre prenom: ";
cin.getline(nom_prenom, sizeof(nom_prenom));
cout << "Bonjour, " << nom_prenom;
```

Le nom et le prénom sont enregistrés dans **nom_prenom**, qui peux contenir des espaces. Notez que **cin.getline()**, contrairement à **cin**, ne permet de lire que des chaînes de caractères.

Programme d'exemple

Nous allons créer un petit programme d'exemple : L'utilisateur devra entrer une chaîne de caractère. Les lettres qui seront en minuscule seront passées en majuscule en inversement. Les chiffres ne seront pas modifiés. Le programme ne traitera pas les caractères accentués car ils ne sont pas affichés sous DOS.

```
#include <conio.h>

#include <iostream>

using namespace std;

int main()
{
  char chaine[255];
  int i = 0;

  cout << "Entrez une chaine de caracteres:\n";
  cin.getline(chaine, sizeof(chaine));

  // tant que l'on a pas atteint le '\0' final
  while(chaine[i] != '\0')
  {
```

```
    // s'il s'agit de A --> Z
    if(chaine[i] >= 65 && chaine[i] <= 90)
      // passage en minuscule
      chaine[i] += 32;
    // s'il s'agit de a --> z
    else if(chaine[i] >= 97 && chaine[i] <= 122)
      // passage en majuscule
      chaine[i] -= 32;

    i++;
  }

  cout << "\nChaine modifiee:\n" << chaine << endl;

  cout << "Appuyez sur une touche pour terminer le programme...\n";
  getch();
}
```

Fonction de cstring et de cctype

Il existe des fonction C++ qui sont adaptées à la gestion des chaînes de caractères et de caractères. Ces fonctions sont déclarées dans <*cstring*> ainsi que dans <*cctype*>. Nous allons en voir quelques une.

Nous allons commencer par voir les fonctions de <*cstring*>.

strcat

```
char *strcat(char *chaine1, const char *chaine2) ;
```

La fonction *strcat* concatène *chaine2* à *chaine1* et ajoute le '\0' de terminaison à la fin de *chaine1*. La fonction retourne *chaine1*.

```
char chaine1[32] = "Hello ";
strcat(chaine1, "world!!!"); // renvoie "Hello world!!! " (aussi contenu dans chaine1)
```

strchr

```
char *strchr(const char *chaine, int ch) ;
```

La fonction *strchr* retourne un pointeur (la notion de pointeur sera abordée dans un prochain tutorial) sur la première apparition de *ch* dans *chaine*. La fonction renvoie *null* si *ch* n'est pas trouvé.

strcmp

```
int strcmp(const char *chaine1, const char *chaine2) ;
```

Cette fonction compare les deux chaîne de manière lexicographique. Voici ses valeurs de retour :

- Valeur de retour plus petite que 0 : *chaine1* est inférieure à *chaine2*
- 0 : *chaine1* et *chaine2* sont égales
- Valeur de retour plus grande que 0 : *chaine1* est supérieure à *chaine2*.

strcpy

```
int strcpy(char *chaine1, const char *chaine2) ;
```

Cette fonction copie *chaine2* dans *chaine1*. La chaîne *chaine2* doit être terminée par le caractère '\0'.

strlen

```
size_t strcpy(const char *chaine) ;
```

Cette fonction renvoie le nombre de caractère dont est constitué *chaine*. Le délimiteur de fin de chaîne de *chaine* doit être '\0' qui n'est pas pris en compte lors du calcul de la longueur.

La fonction renvoie un **size_t** qui est une sorte d'**unsigned int**.

strstr

```
char *strchr(const char *chaine, const char *chaine2) ;
```

Retourne un pointeur sur la première apparition de *chaine2* dans *chaine1*.

Voici maintenant quelques fonctions déclarées dans <*cctype*>.

isalnum

```
int isalnum(int ch) ;
```

Renvoie une valeur différente de 0 si *ch* est un caractère alphanumérique (si c'est une lettre ou un chiffre). Retourne 0 si ce n'est pas le cas.

isalpha

```
int isalpha(int ch) ;
```

Renvoie une valeur différente de 0 si *ch* est une lettre. Retourne 0 si ce n'est pas le cas.

isdigit

```
int isdigit(int ch) ;
```

Renvoie une valeur différente de 0 si *ch* est un chiffre, 0 dans tous les autres cas.

islower

```
int islower(int ch) ;
```

Renvoie une valeur différente de 0 si *ch* est une lettre en minuscule, 0 dans tous les autres cas.

ispunct

```
int ispunct(int ch) ;
```

Renvoie une valeur différente de 0 si *ch* est un signe de ponctuation, 0 dans tous les autres cas.

isspace

```
int isspace(int ch) ;
```

Renvoie une valeur non-nulle si *ch* est un caractère d'espacement (tabulation, espace, saut de page, retour chariot), 0 dans tous les autres cas.

isupper

```
int isupper(int ch) ;
```

Renvoie une valeur différente de 0 si *ch* est une lettre en majuscule, 0 dans tous les autres cas.

tolower

```
int tolower(int ch) ;
```

Cette fonction renvoie la minuscule équivalente à *ch* le retour est *ch* dans le cas ou *ch* est déjà une minuscule ou un autre caractère.

toupper

```
int toupper(int ch) ;
```

Cette fonction renvoie la majuscule équivalente à *ch* le retour est *ch* dans le cas ou *ch* est déjà une majuscule ou un autre caractère.

Voici la fin de se tutorial. Familiarisez-vous bien avec les chaînes de caractères car elles sont un élément important de la programmation.

Le prochain tutorial portera sur les fonctions.