Les tableaux

Un tableau est une **variable "multiple"** permettant de gérer des données en mémoire,
donc **volatiles**

=> si on éteint la machine, ces données sont perdues.

Ne pas confondre un tableau et un fichier (données pérennes)

Toutes les données d'un tableau ont même type.

Les données ont le même nom, et ne sont différenciées que
par leur indice de "rangement" dans le tableau

Les indices commencent à 0

La notion de pointeur, est intimement liée à celle de tableau

Exemple

Tableau "numeros_loto" comportant 10 éléments.

Si on définit la taille du tableau lors de sa déclaration, on écrira :

type numeros_loto[10];

Le 1er élement d'un tableau est : **numeros_loto[0]**

Le dernier élément est noté **numeros_loto[9]**

Le nombre d'éléments d'un tableau peut être fixé à la déclaration de celui-ci,
mais ce n'est pas obligatoire.

Les indices d'accès à un tableau ne sont pas contrôlés, et un mauvais accès pourra donner :

- n'importe quoi (résultat imprévu et imprévisible)
- Bus Error
- Segmentation Fault (tentative d'accès à un emplacement interdit)

Tableaux a 1 dimension

Syntaxe : ***type*** **identificateur[*valeur*]**

Exemples :

```
/* mon premier tableau */

#include <stdio.h>
#include <stdlib.h>

#define TAILLE_TABLEAU          96

short int un_tableau[TAILLE_TABLEAU] ;

int main(void)
{
        ....
        for ( short i = 0 ; i < TAILLE_TABLEAU ; i++)
        {
                /* on initialise le tableau */
                un_tableau[i] = 0 ;
        }

        for (short j = 0 ; j < TAILLE_TABLEAU ; j++ )
        {
                fprintf( stdout, "Element numero %hd = %hd \n" ,  \
                                        j, un_tableau[j] );
        }
        ....

        return EXIT_SUCCESS ;
}
```

Autre initialisation possible :

/* Declaration explicite d'un tableau de 8 elements, *
 *et initialisation EXPLICITE de chaque element */

long couleurs[10] = {123, 3, 45, 123456 , 5, 6 , 7 , 8, 9, 0};

/* Declaration d'un tableau dont la taille est déterminée par le nombre d'éléments *
 * qui seront explicitement initialises. */

long prix[ ] = { 123, 345, 567, 876};

/* on peut aussi ne declare que les premiers elements, le reste sera initialise a 0 */

short numeros_bus[100] = { 1, 2, 3, 4, 5, 78 };

-> à vérifier en TP

Remarque:

numeros_bus[0] contient 1 , couleurs[2] contient 45

Chaines de caractères

Une chaine de caractères **EST** un tableau de caractères,
dont le dernier caractère est toujours le caractère "**NULL**"
et que l'on note encore **\0**

! si on declare :

ville="Paris"

-> on a 6 éléments en tout : 5 lettres + \0

Tableaux de dimension > 1

syntaxe :

type identificateur[valeur_1] ... [valeur_n] ;

Exemple :

```
#define DIM_A  4
#define DIM_B 5

long int tab[DIM_A ] [ DIM_B] ;
```

-> **réserve 20 emplacements de 4 octets en mémoire,**
que l'on peut " visualiser " par :

tab[0][0]	tab[0][1]	tab[0][2]	tab[0][3]	tab[0][4]
tab[1][0]				
			tab[2][3]	
			tab[3][3]	tab[3][4]

On peut ainsi lire le contenu de l'élément de la ligne 3 et de la colonne 4
en lisant le contenu de tableau[2][3]

Dimension > 1 : passage en argument à une fonction

```
void  display_array( int content [ ] [50] ,  long dim_A)

{
        long i ;
        long j ;
        for ( i = 0 ; i < dim_A ; i++ )
        {
                for ( j = 0 ; j < 50 ; j++ )
                {
                        fprintf (stdout, "Value  : %d \n", content[i][j] ) ;
                }
        }
}


int my_array[100][50] ;

/* function call */

display_array( my_array, 30 ) ;
```

Important : on doit toujours spécifier la dimension du tableau pour les dimensions supérieures à la première

Exemples avec du code :

Exemple 1:

```
#include <stdio.h>
#include <stdlib.h>

int tab2[5]={0,1,2,34,5};

int
main (void)
{
    int i;
    for (i=0;i<5;i++)
      {
      fprintf(stdout, " element %d : %d \n", i+1 , tab2[i]);
      }
    return EXIT_SUCCESS;
}
```

Exemple 2 :

```
#include <stdio.h>
#include <stdlib.h>

int tab2[5]={0,1,2,34,5};

int
main (void)
{
    int i;
    for (i=0;i<20;i++)
      {
         fprintf(stdout, "Element %d : %d \n", i+1 , tab2[i]);
      }
    return EXIT_SUCCESS;
}
```

Exemple 3 :

```
#include <stdio.h>
#include <stdlib.h>

int tab2[5]={0,1,2,34,5};
int tab[5];

int
main (void)
{
    int i;
    for (i=0;i<5;i++)
      {
      printf(" element %d : %d \n", i+1 , tab2[i]);
      }

    int val;
    for (i=0;i<5;i++)
      {
      val=tab2[i];
      tab[i]=val;
      printf(" element %d du second tableau est %d \n", i+1, tab[i]*2);
      }
      return EXIT_SUCCESS;
}
```

Exemple 4 :

```
#include <stdio.h>
#include <stdlib.h>

#define NB_ELEMENT      10

int
main(void)
{
      int i;
      float vecteur[NB_ELEMENT];
      float moyenne = 0;
      float somme = 0;
      /* Initialisation du tableau + E/S */
      for (i = 0; i < NB_ELEMENT; i++)
        {
            vecteur[i] = i/10.0;
            fprintf(stdout, "vecteur[%d] = %.3f\n", i, vecteur[i]);
        }

      /* Calcul de la moyenne */
      for (i = 0; i < NB_ELEMENT; i++)
        {
            somme += vecteur[i];    /* Accumulation */
        }
      moyenne = somme / NB_ELEMENT;       /* Division */

      fprintf(stdout, "La moyenne est %.3f\n", moyenne);
      return EXIT_SUCCESS;
}
```

Exemple 5

```
#include <stdio.h>
#include <stdlib.h>
#define NB_ELEMENT      10

int
main(void)
{
      int i;
      float vecteur[NB_ELEMENT];
      float moyenne;
      float variance;
      float diff;
      /* Initialisation du tableau et affichage */
      for (i = 0; i < NB_ELEMENT; i++)
        {
            vecteur[i] = i/10;
            fprintf(stdout, "vecteur[%d] = %.3f\n", i, vecteur[i]);
        }
      /* Calcul de la moyenne */
      moyenne = 0;                                    /* Initialisation */
      for (i = 0; i < NB_ELEMENT; i++)
        {
            moyenne += vecteur[i];                /* Accumulation */
        }
      moyenne /= NB_ELEMENT;                      /* Division */

      /* Calcul de la variance */
      variance = 0;                                   /* Initialisation */
      for(i = 0; i < NB_ELEMENT; i++)
        {
            diff = vecteur[i]-moyenne;
            variance += diff*diff;                /* Accumulation */
        }
      variance /= NB_ELEMENT;                     /* Division */
```

```
	/* Affichage */
	printf("La moyenne est %.3f\n", moyenne);
	printf("La variance est %.3f\n", variance);
	return EXIT_SUCCESS;
}
```

Exemple 6 :

```
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>

#define NB_ELEMENT      6

int
main(void)
{
      int vecteur_initial[] = {3,6,1,9,2,5};
      int vecteur_trie[NB_ELEMENT];
      int min_index;
      int i;
      int j;

      /* Affichage du vecteur initial */
      for (i = 0; i < NB_ELEMENT; i++)
        {
            fprintf(stdout, "vecteur_initial[%d] = %d\n", i, vecteur_initial[i]);
        }
```

```
/* Boucle principale qui remplit le vecteur trie. */
for (i = 0; i < NB_ELEMENT; i++)
  {

      /* La boucle suivante recherche le plus petit element dans       *
       *  vecteur_initial.     Au depart de la boucle on suppose que le plus *
       *  petit element est a l'indice 0, ensuite la boucle parcourt tous les      *
       *  elements depuis l'indice 1 et compare a chaque etape la valeur de  *
       * l'element courant avec celle du plus petit element connu.  Si *
       * l'element courant est plus petit alors on note son indice dans      *
       * min_index et il devient ainsi le plus petit element connu.          */

      min_index = 0;
      for (j = 1; j < NB_ELEMENT; j++)
        {
            if (vecteur_initial[j] < vecteur_initial[min_index])
              {
                  /* un element plus petit a ete trouve on note donc son indice
                     dans min_index
                  */
                min_index = j;
                }
        }

      /* Le plus petit element a ete trouve, on le met a la ieme position de
         vecteur_trie. */
      vecteur_trie[i] = vecteur_initial[min_index];

      /* Puis on modifie la valeur de cet element dans vecteur_initial pour        *
       * le mettre a une valeur plus grande que toutes les autres. Sans cela,      *
       * lors de la recherche de l'element le plus petit dans vecteur_initial *
       * on retrouverait a nouveau le meme et on finirait par remplir        *
       * vecteur_trie avec des elements tous identiques                */

      vecteur_initial[min_index] = INT_MAX;
  }
```

```
      /* Affichage du resultat */
      for (i = 0; i < NB_ELEMENT; i++)
        {
            printf("vecteur_trie[%d] = %d\n", i, vecteur_trie[i]);
        }
      return EXIT_SUCCESS;
}
```

Exemple 7 :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TRUE            1
#define FALSE           0
#define MAX_SIZE  128
#define NB_NOMS         10

typedef char StringT[MAX_SIZE];

int
main(void)
{
      StringT noms[NB_NOMS];
      int ordreAlphabetique, i;

      /* Entree des noms */
      for (i = 0;  i < NB_NOMS; i++)
        {
            printf("Entrez le nom %d : ", i);
            fgets(noms[i], MAX_SIZE, stdin);
        }

      /* INITIALISATION !! */
      ordreAlphabetique = TRUE;

      /* Boucle principale : comparaisons */
      /* ATTENTION, notez bien le i = 1.  Commencer a i = 0 entrainerait
         l'evaluation de noms[i-1] donc un acces en dehors du tableau. */
      for (i = 1; i < NB_NOMS; i++)
        {
            printf("Comparaison nom %d et nom %d : ", i-1, i);
            if (strcmp(noms[i-1], noms[i]) > 0)
              {
                  printf("L'ordre alphabetique N'est PAS respecte !\n");
                  ordreAlphabetique = FALSE;
```

```
            }
          else
            {
                printf("OK, l'ordre alphabetique est respecte.\n");
            }
      }

    /* Affichage du resultat */
    if (ordreAlphabetique)
      {
          printf("*** OK, la liste est dans l'ordre alphabetique. ***\n");
      }
    else
      {
          printf("*** La liste N'est PAS dans l'ordre alphabetique ! ***\n");
      }
    return EXIT_SUCCESS;
}
```