

Chapitre 8:

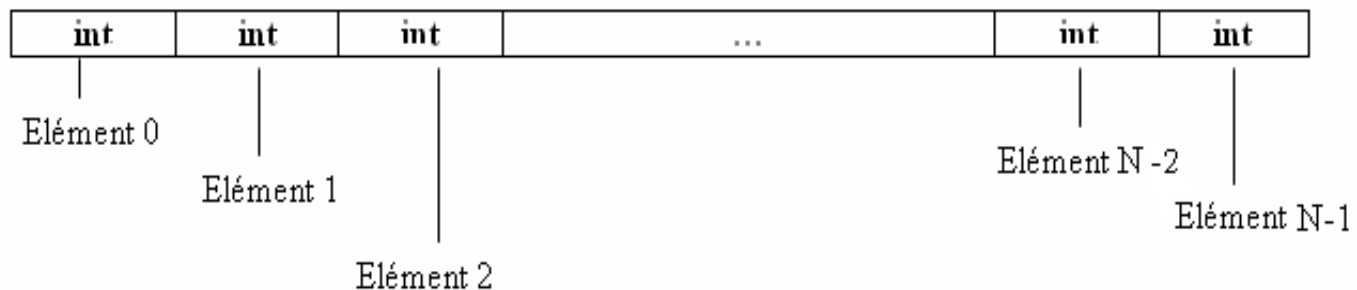
Les tableaux et les pointeurs

Introduction

- Un tableau est un ensemble d'éléments de même type désignés par un identificateur unique. Ces éléments sont rangés en mémoire les uns après les autres. chaque élément est repéré par un indice précisant sa position au sein de l'ensemble.
- Le type d'un tableau peut être n'importe lequel du langage C:
 - ✓ Type élémentaires: char, int, long, float ...
 - ✓ Pointeur (on va détailler ce type plus tard);
 - ✓ Structure (on va détailler ce type plus tard).
- On distingue deux types de tableaux:
 - ✓ Tableaux unidimensionnel (vecteurs);
 - ✓ Tableaux multidimensionnel (tables ou matrices).

Tableaux unidimensionnels

- Un tableau unidimensionnel est composé d'éléments qui ne sont pas eux-mêmes des tableaux.
- On pourrait le considérer comme ayant un nombre fini de colonnes, mais une seule ligne. Par exemple, le tableau suivant est constitué de N éléments de type int:



Tableaux unidimensionnels

- Dans un tableau il faut préciser le type de données, leur nombre, ainsi que le nom sous lequel le programme pourra y accéder a ces éléments.
- La définition d'un tableau unidimensionnel (plus précisément: d'une variable de type tableau) admet la syntaxe suivante:

Type Nom_Du_Tableau[Nombre d'éléments] ;

Tableaux unidimensionnels

- **Type** spécifie le type des éléments du tableau.
 - **Non_Du_Tableau** obéit aux règles régissant les noms de variables.
 - **Nombre d'éléments** est une valeur **constante entière**, qui détermine le nombre d'éléments du tableau.
 - **Les crochets** font partie de la syntaxe.
- **Remarque:** Dans ce cas la taille du tableau est statique (n'est pas variable). Nous verrons plus tard les techniques d'allocation dynamique de mémoire permettent de faire varier la taille des tableaux.

Tableaux unidimensionnels

Exemple

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{ int tab1[6],i;
    /* Lecture des éléments du tableau tab1 */
    for(i=0;i<=5;i++)
    { printf("La saisie de l'\element numero %d\n", i);
      scanf("%d",&tab1[i]);
    }
    printf("Vous avez donne comme tableau \n");
    for(i=0;i<=5;i++)
    { printf("%d ,", tab1[i]);
    }
    printf("\n");
    system("PAUSE");
    return 0;
}
```

Tableaux unidimensionnels

- Initialisation à la définition

On affecte des valeurs initiales aux éléments du tableau dès sa définition. En respectant la syntaxe suivante :

`Type Nom_du_tab[Nbr]={k0,k1,...,kNbr-1};`

Les valeurs `k0,k1,...,kNbr-1` doivent être des constantes.

Tableaux unidimensionnels

Exemple

```
char texte[7]={ 'B', 'o', 'n', 'j', 'o', 'u', 'r' };  
int V [5] = { 1 , 0 , 0 , 0 , 0 } ;  
int V [5] = { 1 } ; // Equivalente à l'instruction précédente
```

Exercices

Créer avec trois façons différentes un tableau à dix éléments de type réels (float) et que vous voulez l'initialiser avec les nombres
 $\{ 0 , 1 , 2 , 3 , 4 , 0 , 0 , 0 , 0 , 0 \}$.

Tableaux à plusieurs dimensions

- Comme tous les langages, C autorise les tableaux à plusieurs dimensions (on dit aussi à plusieurs indices).

- Par exemple la déclaration suivantes:

```
int tab[3][4];
```

réserve un tableau de 12 (3×4) éléments.

- Un élément quelconque de ce tableau se trouve repéré par deux indices comme dans ces notations:

```
tab[2][3];
```

```
tab[i][j];
```

Tableaux à plusieurs dimensions

Exercices

1. Écrire un programme qui lit et qui affiche une matrice de réels de type 3×4
2. Écrire un programme qui lit et qui affiche une matrice de caractères de type 3×4

Tableaux à plusieurs dimensions

Solution

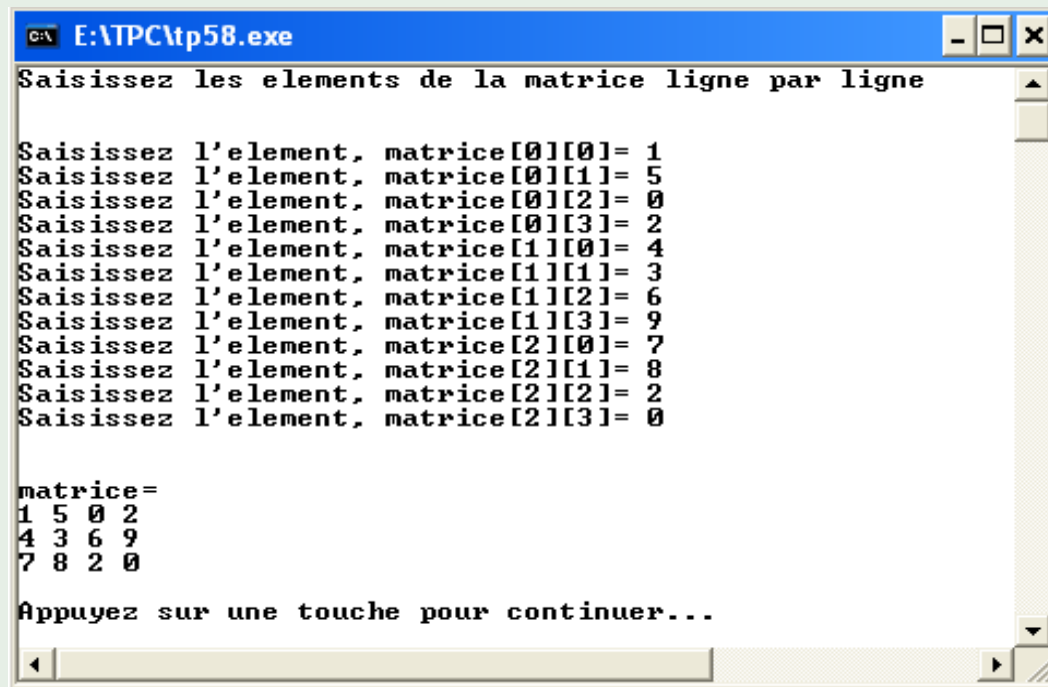
```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int Matrice[3][4];
    int i,j;
    printf("Saisissez les elements de la matrice ligne par ligne\n\n\n");

    for (i=0;i<3;i++)
    {
        for (j=0;j<4;j++)
        { printf("Saisissez l'\element, matrice[%d][%d]= ",i,j);
          scanf("%d",&Matrice[i][j]);
        }
    }
    printf("\n\nmatrice=\n");

    for (i=0;i<3;i++)
    {
        for (j=0;j<4;j++)
        { printf("%d ",Matrice[i][j]);
          }printf("\n");
    }
    printf("\n");
    system("PAUSE");
    return 0;
}
```

Tableaux à plusieurs dimensions

Solution



```
C:\ E:\TPC\tp58.exe
Saisissez les elements de la matrice ligne par ligne

Saisissez l'element, matrice[0][0]= 1
Saisissez l'element, matrice[0][1]= 5
Saisissez l'element, matrice[0][2]= 0
Saisissez l'element, matrice[0][3]= 2
Saisissez l'element, matrice[1][0]= 4
Saisissez l'element, matrice[1][1]= 3
Saisissez l'element, matrice[1][2]= 6
Saisissez l'element, matrice[1][3]= 9
Saisissez l'element, matrice[2][0]= 7
Saisissez l'element, matrice[2][1]= 8
Saisissez l'element, matrice[2][2]= 2
Saisissez l'element, matrice[2][3]= 0

matrice=
1 5 0 2
4 3 6 9
7 8 2 0

Appuyez sur une touche pour continuer...
```

Tableaux à plusieurs dimensions

■ Initialisation à la définition

Voyez deux initialisations équivalentes de la matrice

$$\begin{pmatrix} 1 & 5 & 0 & 2 \\ 4 & 3 & 6 & 9 \\ 7 & 8 & 2 & 0 \end{pmatrix}$$

- ✓ `int Mat[3][4]={ { 1 , 5 , 0 , 2},
 {4 , 3 , 6 , 9},
 {7 , 8 , 2 , 0} };`
- ✓ `int Mat[3][4]={ 1 , 5 , 0 , 2 , 4 , 3 , 6 , 9 , 7 , 8 , 2 , 0 };`

Tableaux à plusieurs dimensions

- La première forme revient à considérer notre tableau comme un tableau formé de trois tableaux de quatre éléments.
- La seconde exploite la manière dont les éléments sont rangés en mémoire et elle se contente d'énumérer les valeurs du tableau suivant cet ordre.
- Là encore, à chacun des deux niveaux, les dernières valeurs peuvent être omises. Les déclarations suivantes sont correctes (mais non équivalentes) :
 - ✓ `int tab [3] [4] = { { 1, 2 } , { 3, 4, 5 } } ;`
 - ✓ `int tab [3] [4] = { 1, 2 , 3, 4, 5 } ;`

Notion de pointeur

- le langage C permet de manipuler des adresses par l'intermédiaire de variables nommées « pointeur ».
- On définit une variable pointeur selon la syntaxe suivante:

`Type * nom_pointeur ;`

- ✓ `nom_pointeur` contient le nom de la variable à définir, qui est dans l'instruction précédente:
`nom_pointeur`
- ✓ `Type` est le type de cette variable pointeur,

Notion de pointeur

- L'indication de type comporte un nouvel élément, l'opérateur `*` que vous connaissez déjà en tant qu'opérateur de multiplication. Cependant dans la définition d'une variable pointeur, cet opérateur a une signification différente « **est un pointeur vers** ».
- Une définition telle que : `int *p` ; peut donc s'interprétée comme:
 - ✓ la variable p est un pointeur vers une donnée de type int.
 - ✓ la variable définie p a le type `int *`.
 - ✓ la variable définie p peut mémoriser l'adresse d'une donnée de type int.

Notion de pointeur

- Considérons l'exemple suivant:

```
int * ad ;  
int n ;  
n = 20 ;  
ad = &n ;  
*ad = 30 ;
```

- L'instruction `int * ad;` réserve une variable nommée `ad` comme étant un « pointeur » vers des entiers.
- L'opérateur `*` désigne le contenu de l'adresse `ad`.
- L'instruction `ad=&n;` affecte à la variable `ad` la valeur de l'adresse de la variable `n`.
- L'instruction `*ad=m;` affecte la valeur de `m` à l'entier ayant pour adresse `ad`.

Notion de pointeur

Exemple

```
int * ad1, * ad2, * ad ;  
int n=10, p=20 ;  
ad1=&n;  
ad2=&p;  
*ad1 = * ad2 + 2;
```

Les variables `ad1`, `ad2` et `ad` sont donc des pointeurs sur des entiers. Remarquez bien la forme de la déclaration, en particulier, si l'on avait écrit :

```
int * ad1, ad2, ad ;
```

La variable `ad1` aurait bien été un pointeur sur un entier (puisque `*ad1` est entier) mais `ad2` et `ad` aurait été, quant à eux des entiers.

Notion de pointeur

- Considérons maintenant ces instructions :

`ad1=&n;`

`ad2=&p;`

`*ad1 = * ad2 + 2;`

- Les deux premières instructions placent dans `ad1` et `ad2` les adresses de `n` et `p` respectivement.
- La troisième instruction place à l'adresse désignée par `ad1` la valeur (entière) d'adresse `ad2`, augmentée de 2.
- Cette instruction joue donc ici le même rôle que :
`n = p + 2 ;`

Notion de pointeur

Exemple

```
#include<stdio.h>
#include<conio.h>
int main()
{ int *ad1, *ad2;
  int m=10,n=20;
  ad1=&m;
  ad2=&n;
  printf("\n l'adresse ou il pointe ad1 est: %d\n",ad1);
  printf("la valeur ou il pointe ad1 est: %d\n",*ad1);
  printf("*****\n\n");
  printf("l'adresse ou il pointe ad2 est: %d\n",ad2);
  printf("la valeur ou il pointe ad2 est: %d\n",*ad2);
  *ad1=*ad2+3;
  printf("\n l'adresse ou il pointe ad1 est: %d\n",ad1);
  printf("la valeur ou il pointe ad1 est: %d\n",*ad1);
  getch();
}
```

Notion de pointeur

Exemple

```
l'adresse ou il pointe ad1 est: 2293612  
la valeur ou il pointe ad1 est: 10  
*****
```

```
l'adresse ou il pointe ad2 est: 2293608  
la valeur ou il pointe ad2 est: 20
```

```
l'adresse ou il pointe ad1 est: 2293612  
la valeur ou il pointe ad1 est: 23
```

-

Incrémentation de pointeur

- Jusqu'ici, nous nous sommes contenté de manipuler, non pas les variables pointeurs elles mêmes, mais les valeurs pointées. Or si une variable `ad` a été déclaré ainsi :

```
int * ad;
```

une expression telle que: `ad+1`; a un sens pour C.

- Pour C, l'expression ci-dessus représente l'adresse de l'entier suivant.
- Dans cet exemple, cela n'a pas d'intérêt car nous ne savons pas avec certitude ce qui se trouve à cet endroit. Mais nous verrons que cela s'avérera fort utile dans le traitement de tableaux ou de chaînes.

Incrémentation de pointeur

- Notez bien qu'il ne faut pas confondre un pointeur avec un nombre entier. En effet, l'expression ci-dessus ne représente pas l'adresse de `ad` augmentée de un (**octet**). Plus précisément, la différence entre `ad+1` et `ad` est ici de **`sizeof(int)`** octets.
- L'opérateur **`sizeof`** fournit la taille en octets d'un type donné.
- Si `ad` avait été déclaré comme double: **`double * ad;`** Cette différence serait de **`sizeof(double)`** octets.

Tableaux et pointeurs

- En langage C, l'identificateur d'un tableau, lorsqu'il est employé seul (sans indice à sa suite) est considéré comme un pointeur (constant) sur le début du tableau. En effet, supposons, par exemple, que l'on effectue la déclaration suivante:
`int t[10];`
- La notation `t` est totalement équivalente à `&t[0]`.
- L'identificateur `t` est considéré comme étant de type pointeur sur le type correspondant aux éléments du tableau, c'est-à-dire, ici, `int *`.

Tableaux et pointeurs

- Voici quelques exemple de notation équivalentes:

$t+1$ $\&t[1]$

$t+i$ $\&t[i]$

$*(t+i)$ $t[i]$

- Pour illustrer ces nouvelles possibilités de notation, voici quelque exemples:

Tableaux et pointeurs

Exemple

```
#include <stdio.h>
#include <conio.h>
int main(int argc, char *argv[])
{
    int i,t[5];
    printf("Saisissez le tableaux t de 5 elements\n");
    for(i=0;i<5;i++)
    {
        scanf("%d", (t+i));
    }
    printf("t = ");
    for(i=0;i<5;i++)
    {
        printf("%d, ", *(t+i));
    }
    printf("\n");
    getch();
}
```

Tableaux et pointeurs

Exemple

Saisissez le tableaux t de 5 elements

```
-2  
3  
0  
9  
-5  
t = -2, 3, 0, 9, -5,
```

Les opérateurs réalisables sur les pointeurs

■ Comparaison

- ✓ On ne peut pas comparer que des pointeurs de même type. Par exemple, voici, en parallèle, deux suites d'instructions réalisant la même action : mise à 1 des 10 éléments du tableau t :

```
int t[10] ;  
int * p ;  
for (p=t ; p<t+10 ; p++)  
  *p = 1 ;
```

```
int t[10] ;  
int i ;  
for (i=0 ; i<10 ; i++)  
  t[i] = 1 ;
```

Les opérateurs réalisables sur les pointeurs

■ Comparaison avec la valeur nulle

- ✓ En principe, on ne peut comparer des pointeurs que s'ils sont de même type. Cependant, tout pointeur (quel que soit son type) peut être comparé à 0. Si p est un pointeur, une instruction comme: `if(p==0) printf(" Erreur \n");` est possible.
- ✓ Dans telle comparaison, on n'utilise pas la constante numérique 0, mais la constante symbolique `NULL` afin de montrer qu'il s'agit d'une comparaison des pointeurs (donc adresses):
`if(p==NULL)`
`printf(" Erreur \n");`
- ✓ La constante `NULL` est définie dans le header `stdio.h`.

Les opérateurs réalisables sur les pointeurs

- **Affectation par un pointeur sur le même type**
Soient P1 et P2 deux pointeurs sur le même type de données, alors l'instruction
P1 = P2;
fait pointer P1 sur le même objet que P2
- **Addition et soustraction d'un nombre entier**
Si P pointe sur l'élément A[i] d'un tableau, alors :

P+j pointe sur A[i+j]

P-j pointe sur A[i-j]

Les opérateurs réalisables sur les pointeurs

■ Incrémentation et décrémentation d'un pointeur

Si P pointe sur l'élément A[i] d'un tableau, alors après l'instruction:

P++; P pointe sur A[i+1]

P+=j; P pointe sur A[i+j]

P--; P pointe sur A[i-1]

P-=j; P pointe sur A[i-j]

■ Domaine des opérations

L'addition, la soustraction, l'incrémentation et la décrémentation sur les pointeurs sont seulement définies à l'intérieur d'un tableau. Si l'adresse formée par le pointeur et l'indice sort du domaine du tableau, alors le résultat n'est pas défini.

Les opérateurs réalisables sur les pointeurs

■ Les pointeurs génériques

✓ Le pointeur:

`void *`

désigne un pointeur sur un objet de type quelconque (on parle souvent de pointeur générique). Il s'agit d'un pointeur sans type.

- ✓ Une variable de type `void *` ne peut pas intervenir dans des opérations arithmétiques.
- ✓ Les pointeurs génériques sont théoriquement compatibles avec tous les autres.

Types de données en C

- Les données d'un programme se répartissent en trois catégories :
 - ✓ Les données **statiques** qui occupent un emplacement parfaitement défini lors de la compilation.
 - ✓ Les données **automatiques** qui sont créées et détruites au fur et à mesure de l'exécution du programme.
 - ✓ Les données **dynamiques** qui sont créées à l'initiative du programmeur.

Type de données en C

- D'une manière générale, l'emploi de données statiques présente certains défauts intrinsèques. Citons deux exemples :
 - ✓ Les données statiques ne permettent pas de définir des tableaux de dimensions variables,
 - ✓ La gestion statique ne se prête pas aisément à la mise en œuvre de listes chaînées, d'arbres binaires,...

Allocation dynamique de mémoire

- Les données dynamiques vont permettre de pallier ces défauts en donnant au programmeur l'opportunité de s'allouer et de libérer de la mémoire dans le « tas », au fur et à mesure de ses besoins.
- Il existe deux principales fonctions C permettant de demander de la mémoire au système d'exploitation et de la lui restituer.
- Elles utilisent toutes les deux des pointeurs génériques. Leur syntaxe est la suivante :
 - ✓ `malloc(taille)`
 - ✓ `free(pointeur)`

Allocation dynamique de la mémoire

- **malloc** (abréviation de « Memory ALLOCation ») alloue de la mémoire. Elle attend comme paramètre la taille de la zone de mémoire à allouer et renvoie un pointeur non typé (**void ***).
- **free** (pour « FREE memory ») libère la mémoire allouée. Elle attend comme paramètre le pointeur sur la zone à libérer et ne renvoie rien.
- Lorsqu'on alloue une variable typée, on doit faire un transtypage du pointeur renvoyé par **malloc** en pointeur de ce type de variable.
- Pour utiliser les fonctions **malloc** et **free**, vous devez mettre au début de votre programme la ligne :
#include <stdlib.h>

Allocation dynamique de la mémoire

Exemple

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int main()
{ float *ad1, *ad2;
  ad1=(float*) malloc(4);
  ad2=(float*) malloc(4);
  *ad1=-45.78;
  *ad2=678.92;
  printf("ad1=%d et ad2=%d\n",ad1,ad2);
  printf("val pointee par ad1=%.2f et val pointee par ad2=%.2f\n",*ad1,*ad2);
  free(ad1);
  free(ad2);
  getch();
}
```

Allocation dynamique de la mémoire

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
main()
{
    int *i;
    i = (int*)malloc(sizeof(int));
    *i = 300;
    printf(" adresse = %d   variable = %d\n",i,*i);
    free(i);
    getch();
}
```

Allocation dynamique de la mémoire

Exemple

```
#include <stdlib.h>
#include <conio.h>
main()
{
    char *adr_deb,c;
    int i,imax,compt_e = 0,compt_sp = 0;
    adr_deb = (char*)malloc(30);
    printf("\nADRESSE DU TEXTE: %d (ATTRIBUEE PAR LE COMPILATEUR)",adr_deb);
    printf("\nENTRER UN TEXTE: ");
    for (i=0;((c=getchar())!='\n');i++) *(adr_deb + i) = c;
    imax = i;
    for (i=0;i<imax;i++)
    { c = *(adr_deb+i);
      printf("\nCARACTERE: %c ADRESSE: %d",c,adr_deb+i);
      if (c=='e') compt_e++;
      if (c==' ') compt_sp++; }
    printf("\nNOMBRE DE e: %2d NOMBRE d'espaces: %2d\n",compt_e,compt_sp);
    free(adr_deb);
    getch();
}
```