STRUCTURES DE DONNÉES
MIP — S4

Pr. K. Abbad & Pr. Ad. Ben Abbou & Pr. A. Zahi
Département Informatique
FSTF
2019-2020

Séance 2

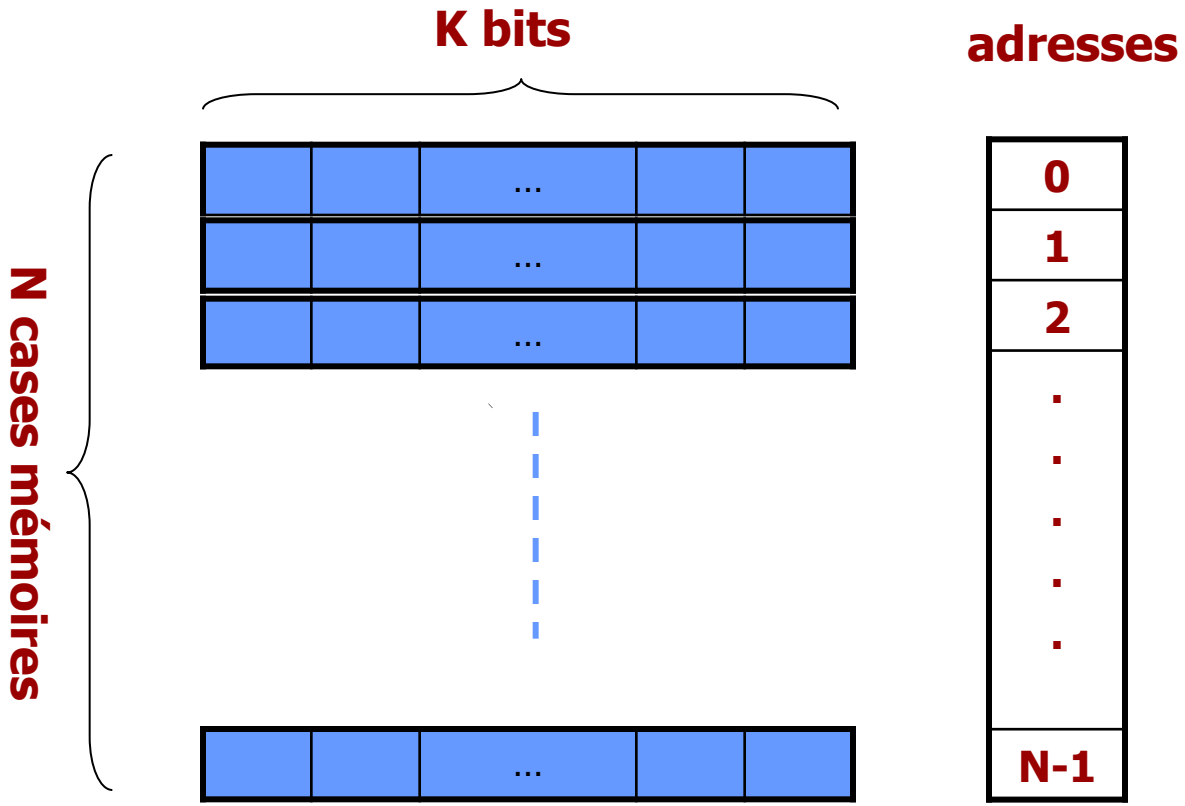
- Allocation dynamique

Allocation dynamique

- Mémoire d'un programme
- Modes d'allocation
- Allocation dynamique

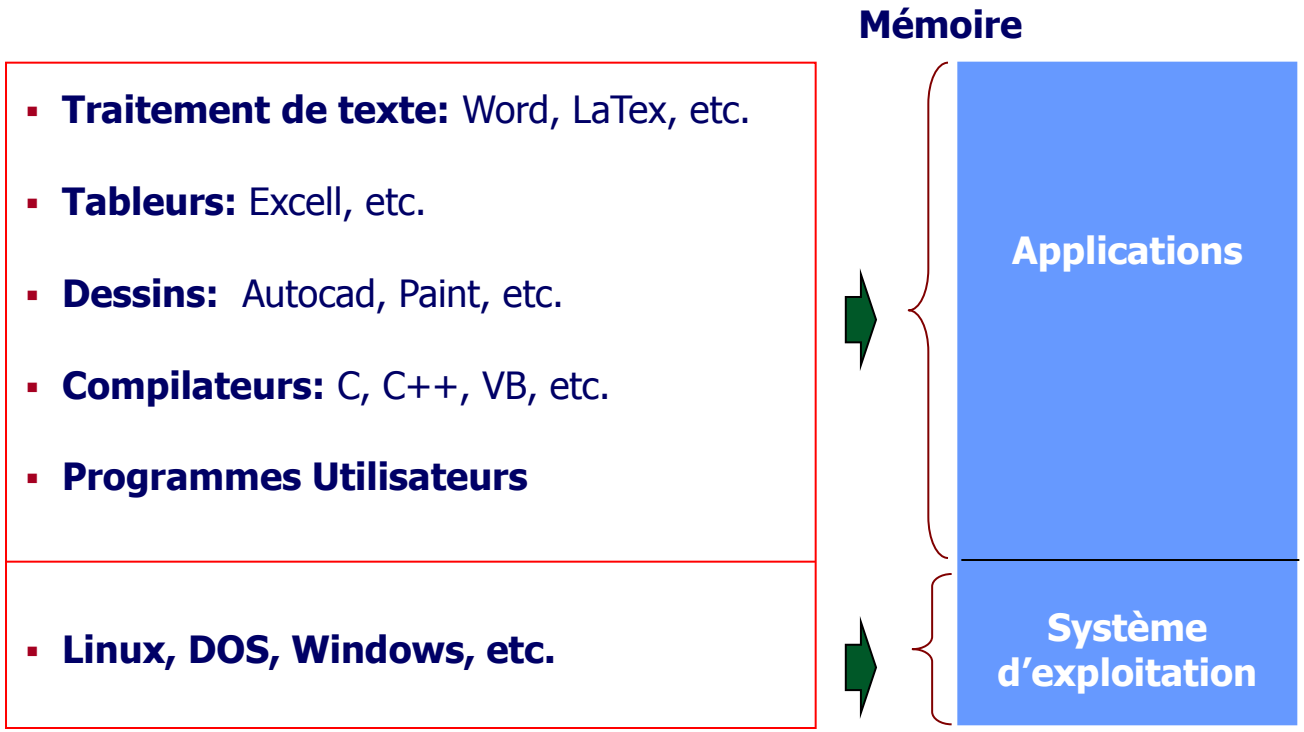
Mémoire de l'ordinateur

- Composée de **N cases mémoires** de **k** bits
- Chaque case est identifiée par une **adresse unique** entre **0** et **N**



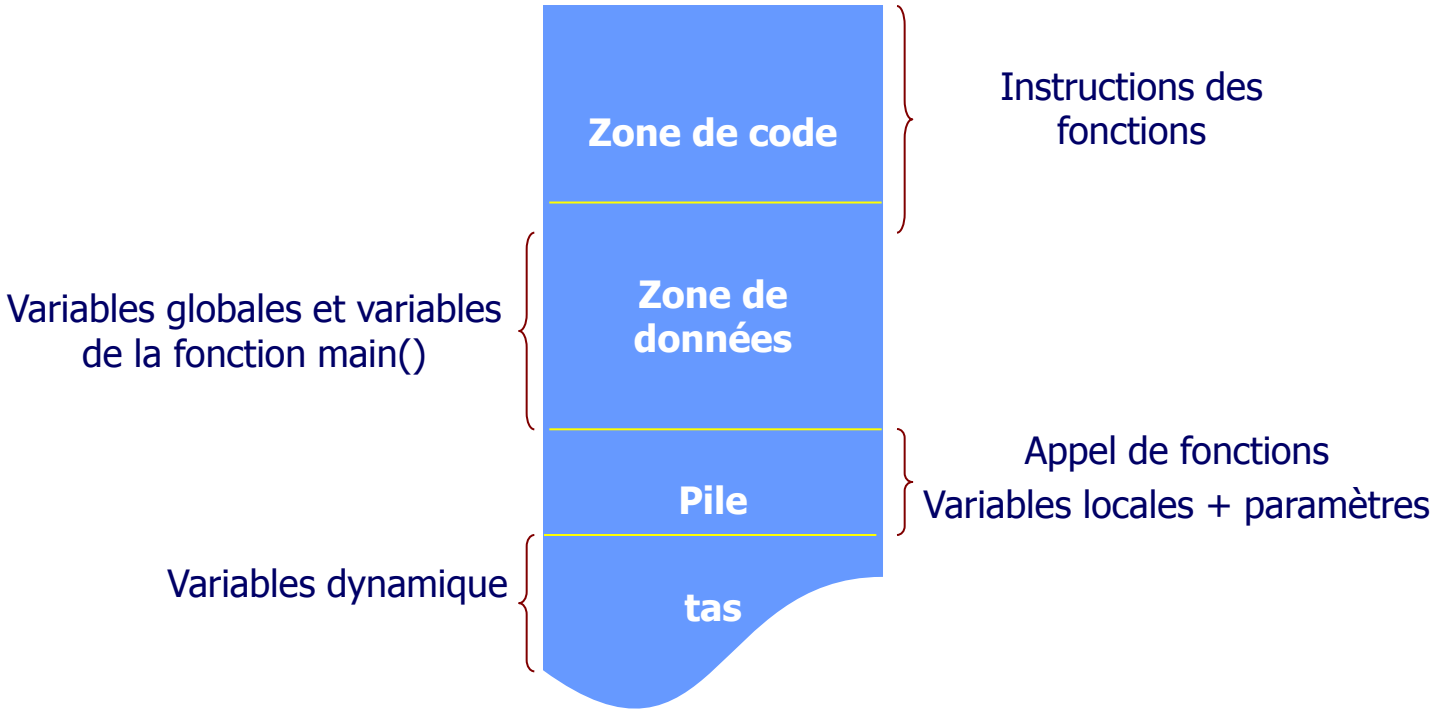
Mémoire de l'ordinateur

- La mémoire de l'ordinateur est divisée en deux parties:
  - **Partie système :** Contient le système d'exploitation actif (DOS, Unix, Linux, Windows, etc.)
  - **Partie application** : Contient les variables, les instructions, appels de fonctions, des applications en exécution.



Mémoire d'un programme

- 4 zones sont réservées pour chaque programme :



- Ces espaces sont alloués à un programme au début de son exécution et sont libérés à la fin de l'exécution.

Modes d'allocation de la mémoire

- La mémoire est attribuée à un programme par le système d'exploitation (Windows, Linux, etc.)
- Deux modes d'allocation sont offerts:
  - Allocation statique

    ⇒ La mémoire est **allouée** et **libérée automatiquement**
  - Allocation dynamique

    ⇒ La mémoire est **allouée** et **libérée** à la **demande du programme (**programmeur**)**

Modes d'allocation —*Allocation statique*

- Permet de réserver de l'espace à toutes les variables du programme.
  - L'allocation s'effectue au début de l'exécution.
  - La libération s'effectue à la fin de l'exécution.
- les variables statiques sont stockées dans la zone de données —*Zone statique*
- les variables dynamiques (de type pointeur) sont stockées dans le tas — *Zone dynamique*

⇨ les adresses attribuées ne peuvent pas changer au cours de l'exécution

Allocation dynamique —*Principe*

- Permet de réserver un espace qui sera manipulé par une variable de type pointeur.
- L'espace pointé est réservé dans la zone statique.
- L'adresse de l'espace réservé est affecté au pointeur concerné.
- L'espace est alloué et libéré au cours de l'exécution à la demande du programmeur

les adresses affectées à un pointeur peuvent changer au cours de l'exécution

Allocation dynamique —*Scénario d'utilisation*

- Déclaration d'un pointeur sur un type de données.

```
nom_type *ptr_X ;
```

- Allocation à l'aide de la fonction *malloc* :

```
int t= sizeof(nom_type) ;
ptr_X= (nom_type* ) malloc(t) ;
```

  - la fonction malloc retourne la valeur NULL en cas d'échec.

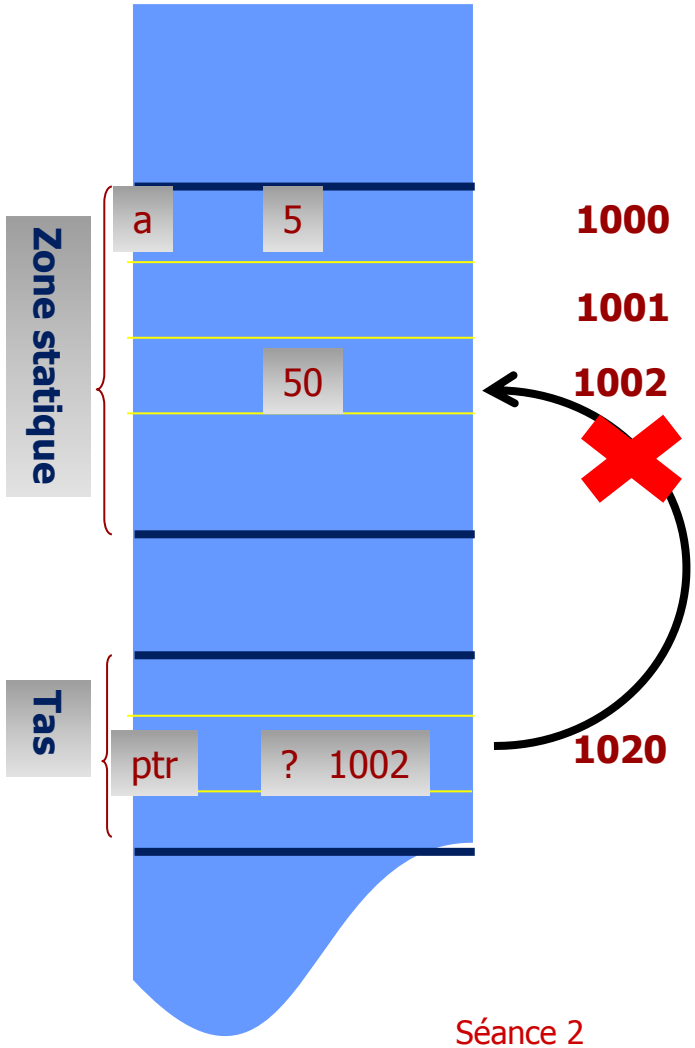
Il faut toujours tester sur l'adresse retournée pour s'assurer qu'elle est valide.

```
if (ptr_X!= NULL) {
/*   …      */ ;
}
```

Allocation dynamique —*Exemple*

```
void main()
{
int a=5;
int *ptr  ;
   /* allocation mémoire pour une
    variable entière X */

ptr= (int*)malloc(siezof(int));
   /* on suppose que la fonction a
   retourné l'adresse 1002*/
   if (ptr_X!= NULL) {
     *ptr = 50;
      free(ptr);
      ptr=NULL;
}
}
```

Current

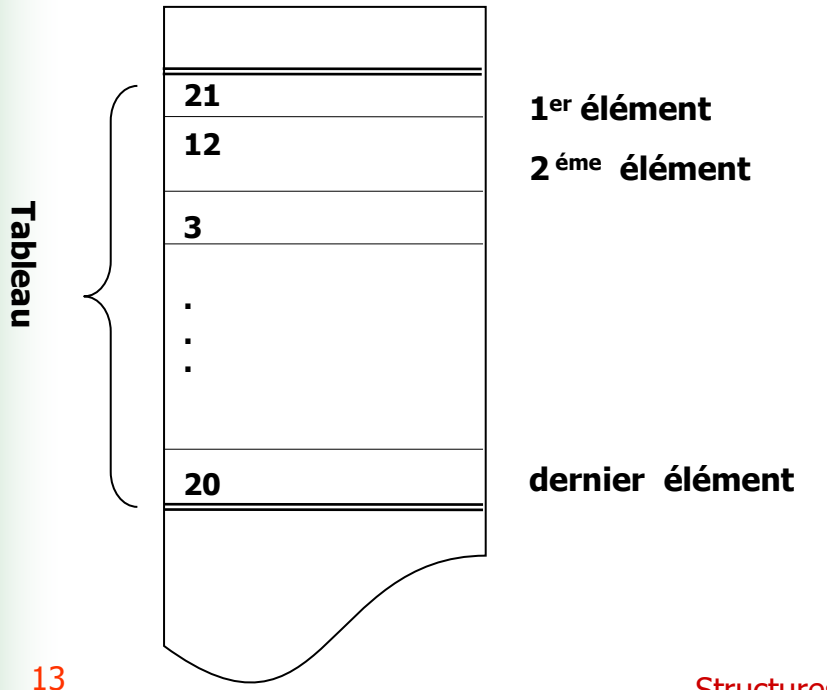


Allocation dynamique —*Remarques*

- L'**adresse** d'un pointeur est attribuée:
  - Statiquement au début du programme
  - Dans la zone dynamique (le tas)
- L'**espace pointé** par un pointeur est alloué:
  - Dynamiquement par le programmeur
  - Dans la zone statique

Allocation dynamique — *Tableaux*

- Le nom d'un tableau est un pointeur qui contient l'adresse du premier élément.
- Représentation contigüe en mémoire.
  - Les éléments sont stockés les un à la suite des autres.



Caractéristiques d'un tableau

- l'adresse du premier élément
- la taille maximale

Allocation dynamique — *Tableaux*

- Deux types sont distingués :

  - Tableau statique
    - Le nom du tableau est pointeur **constant**.
    - Le nombre maximale est spécifié lors de la déclaration.

    nom_type T[CAP], avec CAP est une constante

  - Tableau dynamique
    - Le nom du tableau est un pointeur **variable**.
    - Le nom du tableau et le nombre maximale sont spécifiés lors de l'**allocation** à la demande du programmeur.

    nom_type * T;

Allocation dynamique — *Tableaux*

- l'espace requis par un tableau dynamique est alloué au cours de l'exécution du programme.
- Lors de l'allocation, Il faut préciser les informations suivantes:
  - Le type des éléments
  - La taille d'un élément
  - Le nombre maximale d'éléments
- Plusieurs fonctions d'allocation
  - malloc
  - calloc
  - realloc

Tableaux dynamiques — *malloc*

- Objectif — allouer un bloc contigüe de mémoire pour stocker un tableau.

➡ Il est recommandé de définir une fonction d'allocation qui:

  - Prend en entrée la taille maximale du tableau.
  - Fournit un pointeur sur le début de la zone contigüe allouée — *adresse du premier élément*.

➡ Deux façons de faire :

  - L'adresse de la zone allouée est retournée par la fonction.
  - L'adresse de la zone allouée est passée comme paramètre à la fonction.

Tableaux dynamiques — *malloc*

- L'adresse de la zone allouée est retournée par la fonction.

```
typ_elem* allouer_tab_1(int taille)
{

  type_elem* tab;
  tab = (type_elem*) malloc(taille*sizeof(type_elem)) ;

  return tab;
 }
```

Tableaux dynamiques — *malloc*

- L'adresse de la zone allouée est passée comme paramètre à la fonction

  ➡ On doit passer à la fonction l'adresse du pointeur du premier élément.

```
void allouer_tab_2(int taille, type_elem** tab)
{

  *tab = (type_elem*) malloc(taille*sizeof(type_elem)) ;

 }
```

Tableaux dynamiques — *malloc*

- Exemple

```
void main(){

          int *tab1   ;  int *tab2   ;

/*allocation de l'espace à un tableau de 10 entiers à l'aide la
    fonction1 */
    tab1 = allouer_tab_1(10);
    if (tab1 != NULL) {
      /* utiliser tab1*/
         ...   }

/*allocation de l'espace à un tableau de 10 entiers à l'aide la
    fonction2 */
     allouer_tab_2(10,&tab2);
    if (tab2!= NULL) {
      /* utiliser tab2*/
         ...   }
              ...
    }
```

Tableaux dynamiques — *malloc*

- Exercice

Généralisez la première fonction pour allouer de l'espace à un tableau de n'importe quel type.

```
void* allouer_tab_1(int taille) {
  void* tab;
  tab = (void*) malloc(taille*sizeof(void)) ;

     return tab;
 }
```

Tableaux dynamiques — *malloc*

- Généralisation de l'allocation —*Exemple*

```
/*définition de la structure complexe */
    typedef struct {
        double pi;
         double pr;
    } complexe;

void main(){

    int *tab_int ;  double*tab_double;
    char *tab_ch ;  complexe *tab_comp;

    tab_int =(int*) allouer_tab_1(10);
    tab_comp = (complexe*)allouer_tab_1(10);
    tab_double = (double*) allouer_tab_1(10);
    tab_ch = (char*) allouer_tab_1(10);

      ...
    }
```

Tableaux dynamiques — *calloc*

- Objectif — allouer un bloc contigüe de mémoire pour un tableau et l'initialise par 0.
  - Prend en entrée la taille maximale du tableau.
  - Fournit un pointeur sur le début de la zone contigüe allouée — *adresse du premier élément*.
- La fonction d'allocation définie par l'utilisateur n'a besoin que de la taille.

```
type_elem* allouer_tab_init(int taille) {

  type_elem* tab;
  tab = (type_elem*) calloc(taille, sizeof(type_elem)) ;

   return tab;
 }
```

Tableaux dynamiques — *calloc*

- Généralisation à n'importe quel type à l'aide du type void*

  ➡ La fonction a besoin de la taille d'un élément du tableau

```
void* allouer_tab_init(int taille, int taille_elem) {

 void * tab;
  tab = (void *) calloc(taille, taille_elem) ;

   return tab;
 }
```

Tableaux dynamiques — *calloc*

- Généralisation de l'allocation —*Exemple*

```
/*définition de la structure complexe */
    typedef struct {
       double pi;
        double pr;
    } complexe;

void main(){

int *tab_int ;  double*tab_double;
char *tab_ch ;  complexe *tab_comp;

 tab_int    = (int*)allouer_tab_init(10, sizeof(int));
 tab_comp   =(complexe*)allouer_tab_init(10,sizeof(complexe));
 tab_double=(double*)allouer_tab_init(10, sizeof(double));
 tab_ch     = (char*)allouer_tab_init(10, sizeof(char));
     ...
}
```

Tableaux dynamiques — *realloc*

- Objectif — réallouer un bloc contigu à un tableau avec une **nouvelle** taille.
  - Prend en entrée le tableau et la nouvelle taille.
  - Fournit un pointeur sur le début de la zone contigüe allouée — *adresse du premier élément*

```
type_elem* reallouer_tab(type_elem* tab, int taille) {

tab = (type_elem *) realloc(tab, taille) ;

  return tab;
 }
```

Tableaux dynamiques — *realloc*

- Les données sont conservées ou tronquées si la taille a diminué.

  ➡ Les données sont recopiées après la nouvelle allocation.

- Utile dans les fonctions de mise à jour d'un tableau:

  ➡ Ajout et suppression d'un élément;

Tableaux dynamiques — *realloc*

- Exemple — *ajout d'un élément dans un tableau d'entiers.*

```
int CAP 10; /* taille maximale du tableau*/

void ajout(int **tab, int* taille, int val){

if (*taille == CAP) CAP = CAP*2;
   *tab = (int*) reallouer_tab(*tab, CAP) ;

 if (*tab != NULL) {

    *tab[*taille] = val;
    *taille = *taille + 1;

   }

}
```

Tableaux dynamiques — *free*

- Syntaxe

```
free(tab);
```

- Libère un espace pointé par le nom d'un tableau dynamique.
- Il ne faut pas utiliser le nom d'un tableau dynamique déjà libéré par free.
- Celui qui alloue est celui qui libère.

Si l'allocation est effectuée à l'aide d'une fonction définie par le programmeur alors il faut aussi définir une fonction de libération.

Tableaux à 2 dimensions — *Définitions*

- C'est un tableau de tableaux

  ➡ Un tableau de taille **(L, C)** est un tableau unidimensionnel de taille **L**, dont les éléments sont des tableaux de taille **C**.

- Tableau statique
  - Le nom du tableau est un pointeur **constant**.
  - Les les dimensions L et C sont spécifiées lors de la déclaration.

  ➡ **type_elem T[L][C],** avec L et C sont des constantes.

- Tableau dynamique
  - Le nom du tableau est un pointeur **variable**.
  - Le nom du tableau et les dimensions L et C sont spécifiés lors de l'**allocation** à la demande du programmeur.

  ➡
```
type_elem ** T;
T=(type_elem**) malloc(L*sizeof(type_elem*)) ;
for (i=0; i<L ; i++)
    Tab[i]= =(type_elem*) malloc(C*sizeof(type_elem*)) ;
```

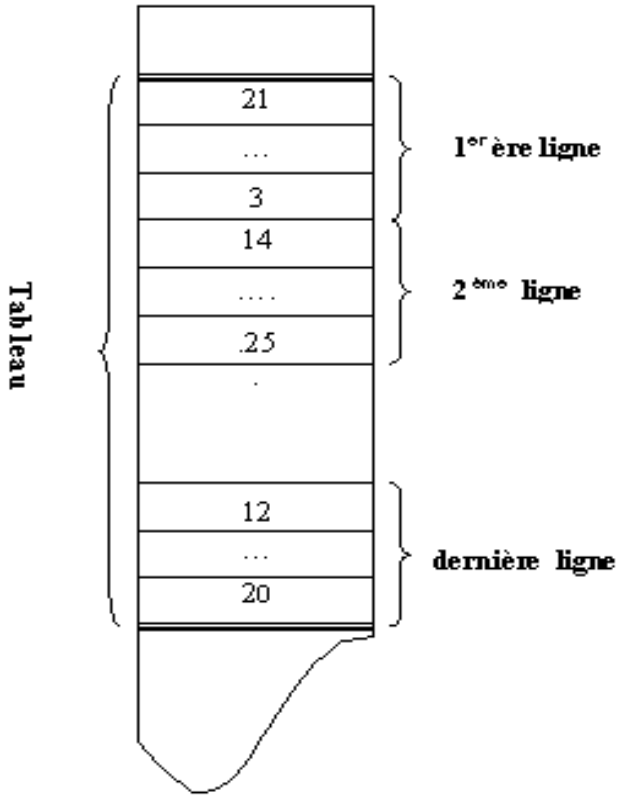
Tableaux à 2 dimensions — *Exemple*

- Pour stocker les notes des 12 modules, d'une section t, on utilise un tableau de dimensions **n** et **12**, où:
  - **n** est le nombre d'étudiants dans la section.
  - Une ligne du tableau représente les notes d'un étudiant dans les 12 modules.
  - Une colonne représente les notes de tous les étudiants dans un module.

	M1	**M2**	**.**	**.**	**.**	**M12**
E1	12	09	.	.	.	16
E2	05	14	.	.	.	12
.						
.						
.						
En	13.5	15	.	.	.	16

Tableaux à 2 dimensions — *Statique*

- En mémoire, les éléments d'un tableau statique sont stockés ligne par ligne.



Tableaux à 2 dimensions — *Allocation*

- Principe — allouer le tableau principal ensuite les sous tableaux.

  - La fonction d'allocation

    - Prend en entrée les deux dimensions.

    - Fournit un pointeur sur le début de la zone allouée — *Un pointeur sur le premier tableau.*

Tableaux à 2 dimensions — *Allocation*

- Méthode 1

```
type_elem** allouer_2tab(int X, int Y) {

int i;
type_elem** tab =(type_elem**) malloc(X*sizeof(type_elem*)) ;

      for (i=0; i<X ; i++)

        Tab[i]=(type_elem*) malloc(Y*sizeof(type_elem))

      return tab;
  }
```

Tableaux à 2 dimensions — *Allocation*

- Méthode 2

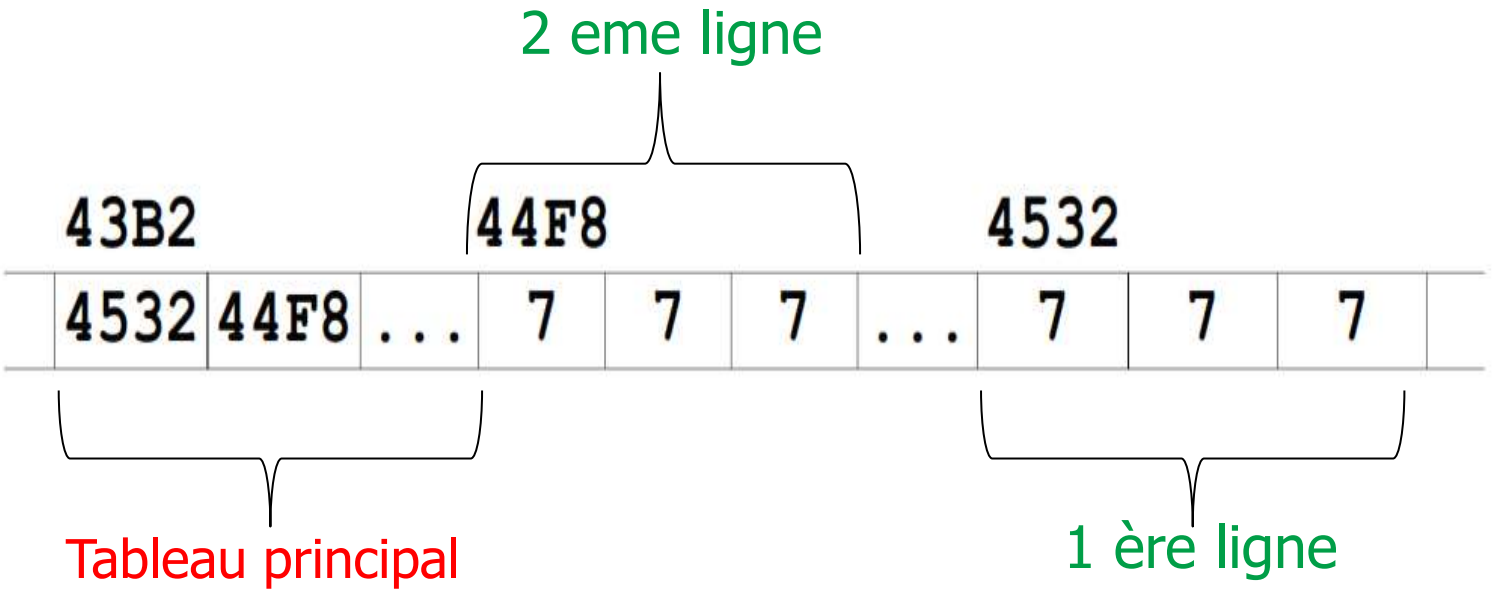
```
typ_elem* allouer_tab_1(int taille) {

  type_elem* tab;
  tab = (type_elem*) malloc(taille*sizeof(type_elem)) ;

  return tab;
 }
```

```
type_elem** allouer_2tab(int X, int Y) {

    int i;
    type_elem** tab =(type_elem**) malloc(X*sizeof(type_elem*)) ;

    for (i=0; i<X ; i++)

        Tab[i]=allouer_tab_1(Y)

    return tab;
  }
```

Tableaux à 2 dimensions — *Allocation*

- Remarques — le tableau principal n'est pas contigu.

  ➡ les sous-tableaux sont distribués et ne sont pas forcément dans l'ordre.



Tableaux à 2 dimensions — *Libération*

- Consiste à libérer les sous tableaux ensuite libérer le tableau principal.
- Prend en entrée le tableau et le nombre de colonnes.

```
void liber_2tab(int **t, int X) {

     int i;
     for (i=0; i<X ; i++)

       if (t[i] != NULL)
          free(t[i]);

     free(t);
 }
```