Algorithmique et programmation

Chapitre 8:Tableaux

1. Tableaux 1D
2. Tableaux 2D

Tableaux

- **Définition**:

Un **tableau** est un ensemble d'éléments (**taille**) de même **type** (stockés de manière contiguë en mémoire) désignés par un **identificateur** unique ; chaque élément est repéré par un **indice** précisant sa position au sein de l'ensemble**.**

- **Caractéristiques:**
  - Identificateur
  - Taille
  - Indice
  - Type

Tableaux (suite)

- **Types:**

Il existe deus types de tableaux: les ***tableaux statiques***, dont la taille est connue à la compilation, et les ***tableaux dynamiques***, dont la taille est connue à l'exécution.

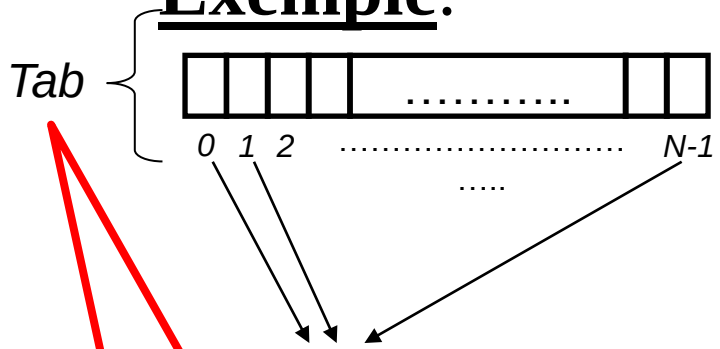
- **Exemples**:

```
/*Tableau statique*/
int  Tab[10];
```

```
/*Tableau dynamique*/
int  *Tab,d;
 d=10;
 Tab=malloc(d*sizeof(int));
 …
 free(Tab);
```

Tableaux 1D

- **Définition**: un tableau **unidimensionnel** (vecteur) est composé d'éléments simples (des éléments qui ne sont pas des tableaux).
- **Exemple**:



- Identificateur du tableau: ***Tab***
- Taille du tableau ***Tab***: ***N***
- Indice du premier élément: ***0***
- Indice du dernier élément: ***N-1***
- Contenu de l'élément n°1: ***Tab[0] ou *(Tab+0)***
- Contenu de l'élément n°2: ***Tab[1] ou *(Tab+1)***
- Adresse de ***Tab[i]*** : ***&Tab[i] ou Tab+i***

Le nom du tableau est un pointeur sur le premier élément

Tableaux 1D (suite)

- **Déclaration**: (Tableau statique)

```
type Nom_tableau[taille];
```

- **Exemple**:

```
float Tab[20];

int N[5]={10,13,11,12,10};
```

Tableaux 1D (suite)

- **Déclaration**: (Tableau dynamique)

```
type *Nom_tableau;
 Nom_tableau=(type*)malloc(taille*sizeof(type));
 …/*Ajouter un traitement en cas d'échec*/
```

- **Exemple**:

```
float *Tab;
int taille;
taille=20;
Tab=(float*)malloc(taille*sizeof(float));
 … /*Ajouter un traitement en cas d'echec*/
free(Tab);
```

- **Remarque**: à la fin du traitement, il faut libérer la zone mémoire allouée à l'aide de la fonction **free()**.

Tableaux 1D (suite)

- **Saisie**:

```
/*Tableau: identificateur « Tab » , taille « dim »*/


for(i=0;i<dim;i++)
  {
        printf ("Entrer Elément n°: %d = \n",i+1);
        scanf("%d",&Tab[i]);      /* scanf("%d",(Tab+i)); */
  }
```

Tableaux 1D (suite)

- **Affichage**:

```
/*Tableau: identificateur « Tab » , taille « dim »*/


for(i=0;i<dim;i++)
  {
        printf("%d  ",Tab[i]);      /* printf("%d  ",*(Tab+i)); */
  }
```

Tableaux 1D (suite)

- **Tri: (méthode de sélection)**

```
/*Tableau: identificateur « Tab » , taille « dim »*/
 for(i=0;i<dim-1;i++)
{   id_min=i;    //Considérer
    for(j=i+1;j<dim;j++)  //Rechercher
    {       if(Tab[j]<Tab[id_min])
                    id_min=j;
     }
     if(id_min!=i)   //Permuter
                    {temp=Tab[i];
                    Tab[i]=Tab[id_min];
                     Tab[id_min]=temp;}
 }
```

Tableaux 1D (suite)

- **Recherche simple**:

```
/*Tableau: identificateur « Tab » , taille « dim »*/


 pos=-1;
for(i=0;i<dim;i++)
   {       if(Tab[i]==rech)
                    pos=i;
    }
if(pos!=-1)
          printf("Elément trouvé à la position : %d  ",pos);
else
           printf("Elément introuvable",);
```

Tableaux 1D (suite)

- **Recherche dichotomique**:

```
/*Tableau trié: identificateur « Tab » , taille « dim »*/
  id_min=0;   id_max=dim-1;   pos=-1; arret=0;
 while(id_min<=id_max && arret==0)
    {      id_moy = (id_min+id_max)/2;
           if(rech==Tab[id_moy])     {  pos=id_moy;  arret=1;}
           else {
                 if(rech<Tab[id_moy])     { id_max=id_moy-1;}
                 if(rech>Tab[id_moy])     { id_min=id_moy+1;}
               }
     }
 if(arret==1)    printf("Elément trouvé à la position : %d  \n",pos);
 else            printf("Elément introuvable",);
```

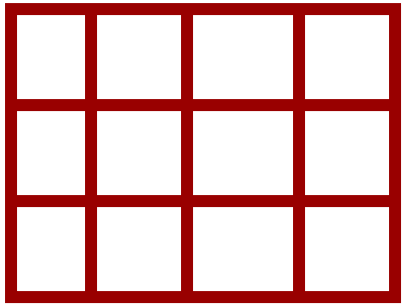
Exercice 1

Un programme en c qui permet de supprimer les éléments identiques consécutifs :

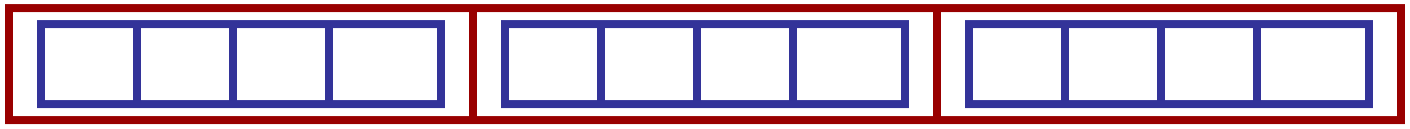
- void lecture(int A[],int dim)
- void affichage(int A[],int dim)
- void suppression(int A[], int *dim)
- main()

Tableaux 2D

- **Définition**: un tableau **bidimensionnel** (matrice) est un tableau de tableaux.
- **Exemple1**: le tableau bidimensionnel suivant (3 lignes, 4 colonnes), est en fait un tableau comportant 3 éléments, chacun d'entre eux étant un tableau de 4 éléments :

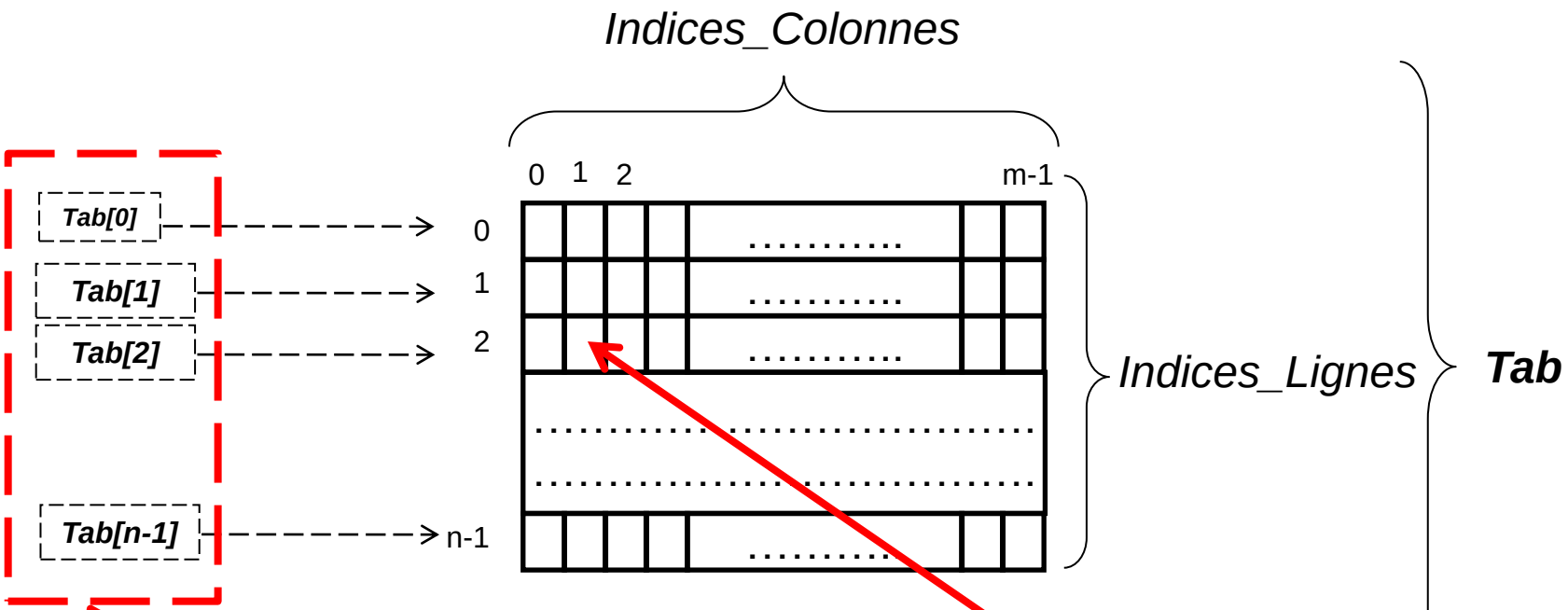


Il est stocké en mémoire de la manière suivante :



Tableaux 2D

- **Exemple2**:



Pour cet élément:

- Adresse: **&Tab[2][1]** ou bien **(Tab[2]+1)** ou bien **(*(Tab+2)+1)**
- Contenu: **Tab[2][1]** ou bien ***(Tab[2]+1)** ou bien ***(*(Tab+2)+1)**

Tableaux 2D (suite)

- **Déclaration**: (Tableau statique)

```
type Nom_tableau[nombre_lignes][nombre_colonnes];
```

- **Exemple**:

```
float Tab[3][4];

float Tab[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

float Tab[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

Tableaux 2D (suite)

- **Déclaration**: (Tableau dynamique)

```
type  **Nom_tableau;
Nom_tableau=(type**)malloc(nombre_lignes*sizeof(type*));
…/*Ajouter un traitement en cas d'échec*/
 for(i=0;i<nombre_lignes;i++)
 {
  Nom_tableau[i]= (type*)malloc(nombre_colonnes*sizeof(type));
   …/*Ajouter un traitement en cas d'échec*/
 }
```

Tableaux 2D (suite)

- **Exemple**: Déclaration et réservation

```
float  **Tab;
int NL,NC;
NL=3; NC=4;
/* Réservation d'un tableau de pointeurs de taille NL*/
Tab=(float**)malloc(NL*sizeof(float*));
 if(Tab==NULL) {printf("Echec\n"); exit(1);}
/* Pour chaque élément du tableau de pointeurs on lui affecte l'adresse du premier élément d'un tableau de taille NC*/
 for(i=0;i<NL;i++)
 {  Tab[i]=(float*)malloc(NC*sizeof(float));
    if(Tab[i]==NULL) {printf("Echec\n"); exit(1);}
  }
```

Tableaux 2D (suite)

- **Exemple (suite)**: Libération de mémoire

```
…/*A la fin du traitement, il faut libérer l'espace mémoire allouée*/

for(i=0;i<NL;i++)
 {  free(Tab[i]); }

free(Tab);
```

Tableaux 2D (suite)

- **Saisie**:

```
/*Tableau: identificateur « Tab » , taille « lig × col »*/

for(i=0;i<lig;i++)
 {
        for(j=0;j<col;j++)
        {
          printf ("Entrer Elément [%d][%d] = \n",i,j);
          scanf("%d",&Tab[i][j]);      /* scanf("%d",(*(Tab+i)+j)); */
        }                              /* scanf("%d",(Tab[i]+j)); */
 }
```

Tableaux 2D (suite)

- **Affichage**:

```
/*Tableau: identificateur « Tab » , taille « lig × col »*/

for(i=0;i<lig;i++)
 {
        for(j=0;j<col;j++)
        {
          printf("%d  ",Tab[i][j]);      /* printf("%d  ",*(*(Tab+i)+j)); */
        }                                /* printf("%d  ",*(Tab[i]+j)); */
        printf("\n");
 }
```

Tableaux 2D (suite)

- **Transposée**:

```
/*Tableau: identificateur « Tab » , taille « lig × col »*/

for(i=0;i<lig;i++)
 {
        for(j=0;j<col;j++)
        {
          Trans[j][i]=Tab[i][j];      /*   *(*(Trans+j)+i)=*(*(Tab+i)+j); */
        }                             /*   *(Trans[j]+i)=*(Tab[i]+j); */
 }
```

Exercice 2

Un programme en c qui permet de calculer la somme de deux matrices A et B (M lignes, L colonnes) :

- int** **allouer**(int L,int C)
- void **liberation**(int **A,int L)
- void **lecture**(int **A,int L,int C)
- void **affichage**(int **A,int L,int C)
- void **somme**(int **A,int **B,int **S,int L,int C)
- void **main()**