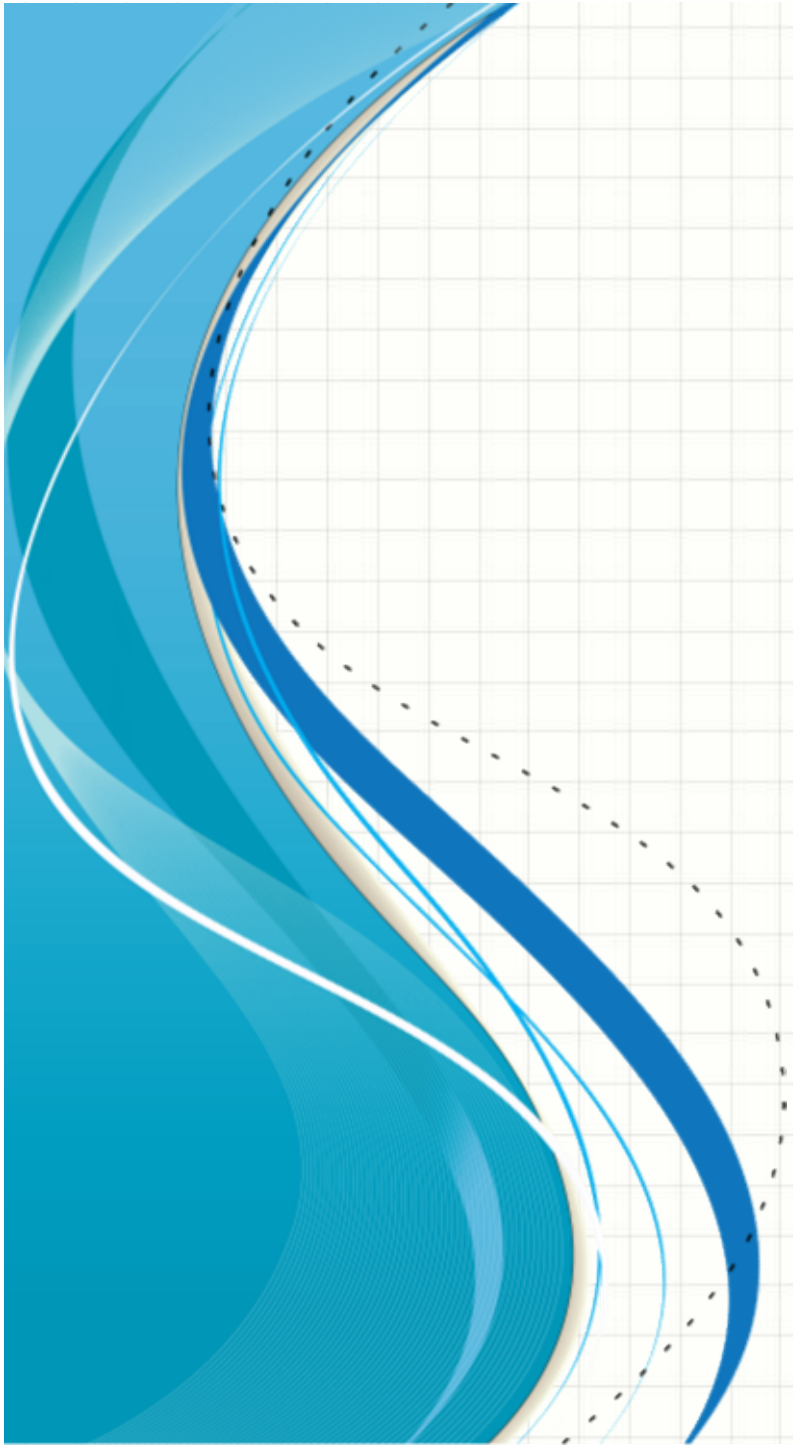


ALGORITHMIQUE AVANCÉE ET PROGRAMMATION

CHAP 2: LES TABLEAUX

Dr. Ikbal Chammakhi Msadaa

Définition

- Un **tableau** est un ensemble d'éléments de même type désignés par un identificateur unique; chaque élément est repéré par un **indice** précisant sa position au sein de l'ensemble.
  - Exemple: supposons que l'on souhaite déterminer, à partir de 20 notes d'élèves (fournies en données), combien d'entre elles sont supérieures à la moyenne de la classe.

Les tableaux à un indice

```
#include <stdio.h>
main()
{   int i, som, nbm ;
    float moy ;
    int t[20] ;

    for (i=0 ; i<20 ; i++)
       { printf ("donnez la note numéro %d : ", i+1) ;
         scanf ("%d", &t[i]) ;
       }
    for (i=0, som=0 ; i<20 ; i++)  som += t[i] ;
    moy = som / 20 ;
    printf ("\n\n moyenne de la classe : %f\n", moy) ;
    for (i=0, nbm=0 ; i<20 ; i++ )
        if (t[i] > moy) nbm++ ;
    printf ("%d élèves ont plus de cette moyenne", nbm) ;
}
```

Les tableaux à un indice

- **Déclarer un tableau**
  - syntaxe : type_elements nom_tableau[nb_cases] ;
  - int t[20];
  - Conventionnellement, la 1ère position prend le numéro 0. Ici, les indices vont de 0 à 19 (de 0 à nb_cases-1).
- **Les éléments d'un tableau**
  - Pour affecter des valeurs dans des cases
    - syntaxe : nom_tableau[numero_case] = valeur ;
    - Exemple: t[2] = 5;
  - Un élément peut apparaître comme opérande d'un opérateur d'incrémentation.
    - exemples: t[3]++ ; --t[i] ;

Les tableaux à un indice

- **Les indices d'un tableau**
  - Un indice peut prendre la forme de n'importe quelle expression arithmétique de type entier (ou caractère, compte tenu des règles de conversion systématique). Par exemple, si n, p, k, j et l sont de type **int**, ces notations sont correctes :
    - t[n-3]
    - t[3*p-2*k+j%l]
  - Il en va de même, si c1 et c2 sont de type **char**, de :
    - t[c1+3]
    - t[c2-c1]

Les tableaux à un indice

- **La dimension d'un tableau**
  - La dimension d'un tableau (son nombre d'éléments) ne peut être qu'une **constante** ou une **expression constante**. Ainsi, cette construction :

```
#define N 50
  .....
int t[N] ;
float h[2*N-1] ;
```

  - est correcte. En revanche, elle ne le serait pas (en C) si N était une constante symbolique définie par const int N=50, les expressions N et 2*N-1 n'étant alors plus calculables par le compilateur (elle sera cependant acceptée en C++).

  - Attention aux bornes du tableau, l'écriture doit se faire entre 0 et nb_cases-1. Si on déclare un tableau de 7 cases (numérotées de 0 à 6) et qu'on tente d'écrire dans la case 9, on obtiendra le message : "Erreur de segmentation" (ou *segmentation fault*).

Les tableaux à un indice

- **Initialisation d'un tableau**
  - syntaxe : type nom_tableau[N] = { val1, val2, ..., valN } ;
  - Exemples:
    - int tab[5] = { 30, 5, 12, 0, 10 } ;
    - int tab[5] = { 30, 5 } ;
    - int tab[5] = { 30, 5, 12 } ;
  - Il est également possible d'omettre la dimension du tableau. Celle-ci sera déterminée par le nombre de valeurs énumérées dans l'initialisation:
  - int tab[] = { 30, 5, 12, 0, 10 } ;

Les tableaux à plusieurs indices

- **Déclarer un tableau**
  - syntaxe : type_elements nom_tableau[taille_dim1][taille_dim2] ;
  - on peut imaginer le tableau sous la forme d'un rectangle avec taille_dim1 qui représente le nombre de lignes et taille_dim2 qui représente le nombre de colonnes.
  - Exemple: int t[5][3];
- **Affecter des valeurs dans des cases**
  - syntaxe : nom_tableau[numero_case_dim1][numero_case_dim2] = valeur ;
  - Si on représente ce tableau sous la forme de lignes et de colonnes, la numérotation des cases s'effectue de **0 à numero_ligne-1** pour les lignes et de **0 à numero_colonne-1** pour les colonnes.

Les tableaux à plusieurs indices

- **Initialisation des tableaux**
  - syntaxe :

    type nom_tableau[N][P] = {val1, val2, ..., valN_P} ;

    ou

    type nom_tableau[N][P] = { {val1_1, val1_2, ..., val1_P}, {val2_1, val2_2, ..., val2_P} ..., {valN_1, valN_2, ..., valN_P}} ;
  - Exemples:

```
int tab [3] [4] = { { 1, 2, 3, 4 } ,
                    { 5, 6, 7, 8 },
                    { 9,10,11,12 }  }


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

Exercices

1. Ecrire un programme qui initialise puis inverse le contenu d'un tableau **Tab** de 10 entiers.
2. Ecrire un programme qui recherche le minimum et maximum dans un tableau **T** de 15 réels saisis au clavier.
3. Ecrire un programme qui permute le contenu de 2 tableaux de 20 entiers saisis au claviers.
4. Ecrire un programme qui calcule et affiche la somme de 2 matrices de taille NxM (avec N et M <= 50). Les éléments des 2 matrices étant saisis au clavier.

Le problème de tri

- On désigne par "**tri**" l'opération consistant à ordonner un ensemble d'éléments en fonction de *clés* sur lesquelles est définie une relation d'ordre.
- Les algorithmes de tri ont une grande importance pratique. Ils sont fondamentaux dans certains domaines, comme l'informatique de gestion où l'on tri de manière quasi-systématique des données avant de les utiliser.
- L'étude du tri est également intéressante en elle-même car il s'agit sans doute du domaine de l'algorithmique qui a été le plus étudié et qui a conduit à des résultats remarquables sur la construction d'algorithmes et l'étude de leur complexité.

Tri à bulles

Cet algorithme parcourt le tableau en comparant 2 cases successives , lorsqu'il trouve qu'elles ne sont pas dans l'ordre souhaité (croissant dans ce cas) , il permute ces 2 cases.
A la fin d'un parcours complet on aura le déplacement du maximum à la fin du tableau . En faisant cet opération N fois , le tableau serait donc trié .

5	1	4	9	8	2
1	5	4	9	8	2
1	4	5	9	8	2
1	4	5	9	8	2
1	4	5	8	9	2
1	4	5	8	2	9

Tri à bulles

```c
#include <stdio.h>
#define N 6
int main()
{
    int t[6]= {5, 1, 4, 9, 8, 2};
    int i, j, temp;
    for (i=0; i<N-1; i++)
    {
        for(j=0; j<N-i-1; j++)
        {
            if(t[j]>t[j+1])
            {
                temp = t[j];
                t[j] = t[j+1];
                t[j+1] = temp;
            }

        }
    }
    for (i=0; i<N; i++)
        printf("%d\t", t[i]);
    return 0;
}
```

5	1	4	9	8	2
1	5	4	9	8	2
1	4	5	9	8	2
1	4	5	9	8	2
1	4	5	8	9	2
1	4	5	8	2	9

Le tri par insertion

- C'est le tri du joueur de cartes. On fait comme si les éléments à trier étaient donnés un par un, le premier élément constituant, à lui tout seul, une liste triée de longueur 1.
- On range ensuite le second élément pour constituer une liste triée de longueur 2, puis on range le troisième élément pour avoir une liste triée de longueur 3 et ainsi de suite...
- Le principe du tri par insertion est donc d'insérer à la $n_{ième}$ itération le $n_{ième}$ élément à la bonne place.

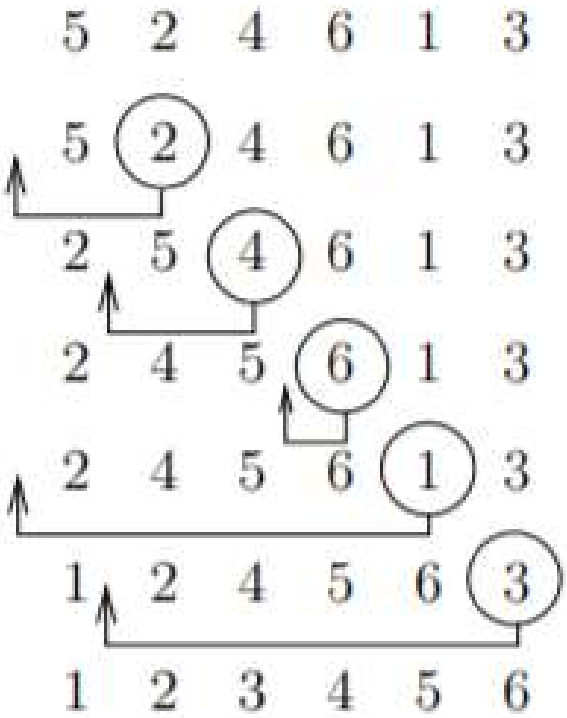
Le tri par insertion

```c
#include <stdio.h>
#define N 6

int main()
{
    int i, j, en_cours, t[N]={5, 2, 4, 6, 1, 3};

    for(i=1; i<N; i++){
        en_cours = t[i];
        for(j=i; j>0 && t[j-1]>en_cours; j--)
            t[j] = t[j-1];

        t[j] = en_cours;
    }
    for(i=0; i<N; i++)
        printf("%d\t", t[i]);
    return 0;
}
```



Recherche Dichotomique

Objectif :

Rechercher une information dans un tableau trié

Méthode :

dichotomique ou « diviser pour régner »

Soit T un tableau de N éléments et val l'élément cherché

- T est trié
- Comparer val avec l'élément du milieu du tableau T.
- Si c'est le même => trouvé
- sinon on recommence sur la première moitié ou la seconde

selon que:

val est < à valmid ou val > valmid

- Arrêt quand l'élément est trouvé ou si fin de tableau

Recherche Dichotomique

```c
#include<stdio.h>
int main()
{
    int T[]={1,2,3,5,6,8,9};

    int iRecherche, iPremier=0, iDernier=6, iMilieu, iTrouve=0;

    printf("Quel élément recherchez-vous ? ");
    scanf("%d",&iRecherche);
    while((iPremier <= iDernier)&&(iTrouve==0))
    {
            iMilieu=(iPremier+iDernier)/2;
            if(T[iMilieu]==iRecherche)
                iTrouve=1;
            else {
               if(T[iMilieu]>iRecherche)
                  iDernier = iMilieu - 1;
               else
                  iPremier = iMilieu + 1;
            }

      }
      if(!iTrouve)
             printf("Cette valeur n'appartient pas à la liste\n");
      else
            printf("Cette valeur appartient à la liste\n");
}
```