

TP1 (suite) : Modularité et Tableaux

1. Les vecteurs :(tableaux à une dimension) :

1.1. Déclaration et utilisation:

La déclaration et l'utilisation des vecteurs dans les sous-programmes (fonctions et procédures) est similaire à leur utilisation dans les programmes principales, sauf si ils ont utilisé comme paramètres formels, on ne doit pas préciser la dimension.

Exemple :

➤ Voici un programme C++ qui lit et affiche un vecteur.

```
#include <iostream>
using namespace std;

//Un module qui permet de remplir un vecteur de n valeurs entières.
void lire_VE (int tab[], int n)
{
    int i;
    for (i=0;i<=n-1; i++)
        cin>>tab[i];
}

//Un module qui permet d'afficher les valeurs d'un vecteur d'entiers.
void Afficher_VE(int tab[], int n)
{
    int i;
    for (i=0;i<=n-1; i++)
        cout <<tab[i];
}

// Programme Principale
int main ()
{
    const int taille_max_tableau = 100 ;
    int taille;
    int v[taille_max_tableau];
    cout << "\n Donnez le nombre d'éléments du vecteur : ";
    cin>>taille;
    cout<< "\n Donnez les éléments du vecteur : \n";
    lire_VE (v, taille);
    cout<< "\n Les elements du vecteur sont : \n";
    Afficher_VE(v,taille);
    system("PAUSE"); //pour stopper la fermeture de la fenêtre
    return 0;
}
```

Ne mettez pas la taille du vecteur ici

Exercice1 :

Ecrire un programme C++ qui permet de lire un vecteur V1 de réelles de taille N et vérifier s'il est trié dans l'ordre croissant ou non.

2. Les matrices :(tableaux à deux dimensions) :

Même chose pour les matrices, la seule différence c'est qu'on doit préciser la deuxième dimension (nombre de colonnes) dans les paramètres formels.

Exemple :

Un programme qui lit et affiche une matrice en utilisant les fonctions :

```
#include <iostream>
using namespace std;
void lire_matrice (int mat[][10],int d1 ,int d2 )
{
  int i,j;
  for(i=0;i<=d1-1;i++)
  for(j=0;j<=d2-1;j++)
  cin>>mat[i][j];
}

void afficher_matrice (int mat[][10],int d1, int d2)
{
  int i,j;
  for(i=0;i<=d1-1;i++)
  {
  for(j=0;j<=d2-1;j++)
  cout<<mat[i][j]<<" ";
  cout<<"\n";
  }
}

int main () {
int m [20][10],n1,n2;

cout<<"\nTapez le nombre de lignes de la matrice : ";
cin>> n1;

cout <<"\nTapez le nombre de colonnes de la matrice : ";
cin>>n2;

cout<<"\nTapez les ellements de la matrice :\n\n";
lire_matrice(m,n1,n2);

cout<<"\n Les ellements de la matrice sont :\n\n";
afficher_matrice (m,n1,n2);

system ("PAUSE"); //pour stopper la fermeture de la fenetre d'exécution
return 0;
}
```

*il faut toujours préciser
la taille dimensions des
colonnes*

Appel à la
fonction
lire_matrice

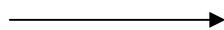
Remarque :

- Les tableaux sont passés toujours par variable, donc on n'a pas besoin de mettre « * » dans les paramètres de sortie.

Exercice 2 :

Ecrire un programme C++ qui lit une matrice et calcule sa transposée ; les lignes de la matrice initiale deviennent les colonnes de la matrice transposée, puis l'affiche.

4	12	33	7
0	7	12	11
6	5	4	13



4	0	6
12	7	5
33	12	4
7	11	13

Exercice 1:

<pre>#include <iostream> using namespace std; //Un module qui permet de remplir un vecteur de n valeurs entières. void lire_VE(int tab[], int n) { int i; for(i=0;i<=n-1; i++) cin>>tab[i]; } // fonction qui vérifie l'ordre des éléments d'un vecteur bool croissant(int tab[], int n) { bool b; int i; b=true; i=1; while ((b==true)&&(i<n)) { if (tab[i]<tab[i-1])b=false; i=i+1; cout<< b<<" ----"; } return (b); }</pre>	<pre>// Programme Principale int main() { const int taille_max_tableau = 100 ; int taille; int v[taille_max_tableau]; cout<<"\n Donnez la nombre d'éléments du vecteur : "; cin>>taille; cout<<"\n Donnez les éléments du vecteur : \n"; //appel à la procédure lire_VE lire_VE(v, taille); //Appel à la fonction croissant if(croissant(v,taille)==1){ cout<<"croissant\n";} else {cout<<"n'est pas croissant\n";} system("PAUSE"); return 0; }</pre>
---	--

Exercice 2 :

<pre>#include <iostream> using namespace std; const int n=2, m=3;// valeur max =10 float M[10][10]; float T[10][10]; int i,j; // la fonction qui permet de lire la matrice void lire_mat (float Mat[][10],int n, int m) { int i; cout<<"\n donnez les valeurs de la matrice initiale :"<<"\n"; for (i=0 ; i<n ; i++) for (j=0 ; j<m ; j++) cin>>M[i][j]; } // la procedure qui calcule le transposé void transpose (float Mat[][10],float tran[][10],int n, int m)</pre>	<pre>// la fonction qui permet d'afficher une matrice void affiche_mat (float Mat[][10],int n, int m) { for (i=0 ; i<n ; i++){ for (j=0 ; j<m ; j++){ cout<<Mat[i][j]<<" "; } cout<<"\n"; } } int main() { //la lecture de la matrice lire_mat(M,n,m); // calcul de la matrice transposée T transpose(M,T,n,m);</pre>
--	--

```
{  
for (i=0 ; i<n ; i++)  
  for (j=0 ; j<m ; j++)  
    tran[j][i]=Mat[i][j];  
}
```

```
// affichage de la matrice initiale  
cout<<"\n la matrice initiale est:"<< "\n";  
affiche_mat(M,n,m);  
// affichage de la matrice transposée  
cout<<"\n la matrice transposee est:"<< "\n";  
affiche_mat(T,m,n);  
system("PAUSE");  
}
```