Série N° 0

Exercice 1 :

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

int main()
{   int N; //taille eff du tab
    Do { cout << "Donner la taille du
                  tableau" ;
         cin >> N ;
       } while ( N<=0) ;
    int Tab[N] ; // la valeur de N
    long som = 0 ;
    for (int i=0 ; i < N ; i++)
    {  cout << "Donner tab[" << i << "]";
       cin >> Tab[i] ;
    } som += Tab[i] ;
    cout << endl;
    int min = tab[0], max = tab[0] ;
    for ( int i = 1 ; i <N ; i++)
    {  if (Tab[i] < min)
           min = Tab[i] ;
       if (Tab[i] > max)
           max = Tab[i] ;
    }
    double moy = (double) som / N ,
    cout << "Les" << N << "éléments de
         ce tableau sont : \n " ;
    for ( int i=0 ; i<N ; i++)
       cout << "Tab[" << i << "]" << Tab[i]
            << '\t' ;
cout << endl;
cout << "la valeur minimale est: " << min << endl;
cout << "la valeur max est:" << max << endl;
cout << "la valeur moy est:" << moy << endl;
return 0,
}
```

Exercice 2 :

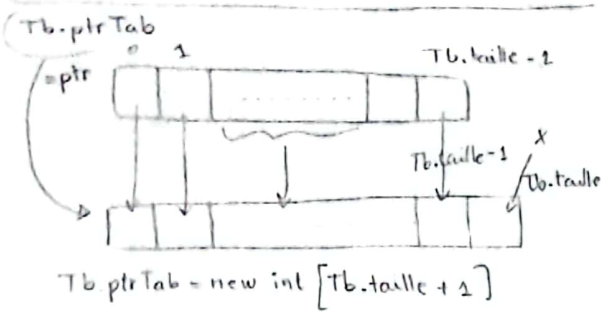
```
#include <iostream>
using namespace std;
typedef struct
 {int *ptrTab;
  int taille ; } Tvecteur ;

// les prototypes des fonctions
void Affichage (Tvecteur &Tab),
void Ajouter (Tvecteur &Tab, int x);
void Supprimer (Tvecteur &Tab, int index),
int main()
{  Tvecteur tb;
   Do { cout << "Préciser la taille dynamique
            à manipuler : ";
        cin >> Tb.taille ;
      } while (Tb.taille <=0) ;
  tb.ptrTab = new int [tb.taille];
  for ( int i=0 ; i<tb.taille ; i++)
    { cout << "Donner tab[" << i << "] : " ;
      cin >> tb.ptrTab [i];
    }
  Affichage (tb) ;
  int a ;
  cout << "Préciser la valeur à ajouter " ;
  cin >> a ;
  Ajouter (tb, a) ;
  Affichage (tb) ;
  int index ;
  cout << "Préciser l'indice de l'element à
       supprimer : " ;
  cin >> index ;
  supprimer (tb, index) ;
  Afficher (tb) ;
     return 0;
}
```

CamScanner

```
void Affichage (Tvecteur &Tab)
{ cout << "Les" << Tab.taille << "elements
        du tableau sont :" << endl;
  for (int i=0 ; i < Tab.taille ; i++)
      cout << Tab.ptrTab [i] << " ";
      cout << endl;
}
```

Tb { ptrTb , taille }



Tb.ptrTab = new int [Tb.taille + 1]

```
void Ajouter ( Tvecteur &Tab , int x )
  { int *ptr =   Tab.ptrTab ;
    Tab.ptrTab = new int (++tab.taille);
    for (int i=0 ; i< Tab.taille - 1 ; i++)
      { Tab.ptrTab [i] = *ptr ;
        ptr ++ ;
      }
    Tab.ptrTab [tab.taille - 1] = X ;
  }
void Supprimer (Tvecteur &Tab, int index)
  { int.ptr = tab.ptrTab ;
    tab.ptrTab = new int [-- tab.taille];
    for ( int i = 0 ; i < index ; i++ )
      { tab.ptrTab[i] = *ptr ;
        ptr ++ ;
      }
    for (int i = index ; i < tab.taille ; i++)
      { tab.ptrTab[i] = *ptr ;
        ptr ++ ;
      }
  }
```

Exercice 3

```
#include <iostream>
#include <cstring>
#include <cstdio>
#define Lmax 100
using namespace std ;
int main ()
 { char chaine [Lmax] ;
   cout << "préciser une ligne de texte" ;
   gets ( chaine) ;
   char *ptr = new char [strlen (chaine)] ;
   for (int i=0 ; i < strlen (chaine) ; i++)
       ptr [i] = chaine [strlen (chaine)-i-1];
   ptr [strlen (chaine)] = '\0';
   puts ( ptr) ;
   int cmp = strcmp ( chaine , ptr) ;
   if ( cmp = = 0)
       cout << "cette est palindrome \n" ;
   else
       cout << "n'est pas palindrome \n",
   return 0 ;
 }
```

CamScanner

Série N° 1

Exercice 1

```
#include <iostream>
using namespace std;
Class CTableau {
    short taille;
    int *ptrValeur;
    void permut (int &a, int &b)
    { int aux; aux=a; a=b; b=aux; }
    Public :
    //les constructeurs
        CTableau ();
        CTableau (short t);
        CTableau (short t, int *ptrtab);
        CTableau( CTableau &Tab);
    //les méthodes publiques
        short GetTaille();
        int *GetValeurs();
        void Tri();
        void Afficher();
        void ChargerTab();
    //Destructeur
        ~CTableau();
};
int main()
{ cout<<"les appels aux constructeur"<<endl;
  CTableau tb0, //déclaration et initialisation
                d'un objet CTableau
  tb0.Afficher();
  cout<<"\n Appel au constructeur 1 arg"<<endl;
  short N; // taille du tableau
  cout<<"Préciser la taille du tableau";
  cin>>N;
  CTableau tb1(N); // tb1 =CTableau(N)
  tb1.Afficher();
  cout<<tb1.GetTaille()<<endl;
  int tab={0,1,2,3,4,5};
  CTableau tb2=CTableau(6,tab);
  tb2.Affiche();
  tb2.Tri();
  tb2.Afficher();
  CTableau tb3=CTableau(tb);
  //CTableau tb3(tb2);
  short N1=tb3.GetTaille();
  int *ptr=tb3.GetValeurs();
  for(int i=0; i<N1; i++)
     cout<<ptr[i]<<"\t";
  cout<<endl;
  /* for(int ptr1=ptr; ptr1<ptr+N1; ptr1++)
        cout<<*ptr1<<"\t";
  */
  return 0;
}
//Définition des méthodes
// les constructeurs
CTableau::CTableau()
{ this->taille = 0;
  ptrValeur = NULL; }
CTableau::CTableau(short t)
{  taille t;
   ptrValeur = new int[t];
   for (int i=0; i<taille;i++)
       ptrValeur[i]=0;
}
CTableau::CTableau(short t, int *ptrtab)
{  this->taille=t;
   this->ptrValeur = new int[t];
   for(int i=0; i<taille; i++)
       ptrValeur[i]=ptrtab[i];
}
CTableau::CTableau(CTableau &Tab)
{ this->taille = tab.taille;
  this->ptrValeur = new int[taille];
  for(int i=0; i<taille; i++)
      this->ptrValeur[i]=tab.ptrValeur[i];
}
```

CamScanner

```
//Destructeur
CTableau :: ~CTableau ()
{   delete [ ] this -> ptr Valeur ; }

//Les méthodes
~~void Afficher~~
void  CTableau :: Afficher ( )
{ cout << "taille de tableau :" << taille << endl;
  for(int i=0; i<taille; i++ )
      cout << ptrValeur[i] << '\t';
  cout << endl; }
short  CTableau :: Gettaille ()
{ return  taille ; }

int *CTableau :: GetValeur ()
{  return (ptrValeur ) ; }

vad  CTableau :: Tri ( )
{ int i,j ;
  for ( i=0 ; i < taille - 1 ; i++ )
     for ( j=i+1 ; j < taille ; j++ )
        if ( ptrValeur [j] < ptrValeur [i])
           permut (ptrValeur [j], ptrValeur[i])
}

void  CTableau :: Chargertab ( )
{ int i ;
  cout << taille << "Elements : \n" ;
  for(i=0 ; i < taille ; i++ )
    { cout << "Donner tab ["<<i<<"]:";
      cin >> ptrValeur [i] ;
    }

~~void~~
```

Série N° 2

Exercice 1

```
#include <iostream>
#include <cstring>
using namespace std,
Class Chaine
{ unsigned taille ;
  char *ptrch,
  Public :
     Chaine ( ) ;
     Chaine (const char * ),
     ~ Chaine ( ),
     unsigned GetTaille ( ) ;
     unsigned Freq_Caractère (char);
    void Eliminer_car (char);
    void Afficher ( ) ;
} ;
int main ( )
{ //initialisation d'un objet chaine
  chaine ch;
  Ch. Afficher ( );
 //initialisation d'un objet chaine avec une
  de chaine de caractère */
  Chaine ch1 ("une ligne de quelque lettre e" );
  ch1. Afficher ( );
 // Elimination d'un caractère précisé
  par l'utilisateur
  cout << "Préciser le caractère à éliminer:" ;
  cin >> car ;
  ch1. Eliminer_Car( car) ;
  ch1. Afficher ( );
  // Elimination de la lettre e
  ch1. Eliminer_car
  ch1. Afficher ( );
  // fréquence de cte lettre
  chaine ch ("ceci une ligne de texte" );
  char car ;
  cout << "Donner un caractère :";
  cin >> car ;
  ch. Afficher( )
  cout << car << "existe" << ch. Freq_Caractère(car)
     << "fois dans cette ligne de texte " ;
  ch. ~Chaine( ) ;
  return 0;
}
  // Définition des méthodes
  Chaine :: Chaine ( )
  { Taille = 0 ;
    ptrch = new char [1],
    *ptrch = '\0' ;
  }
  Chaine :: Chaine (const char *ptrS)
  { Taille = strlen ( ptrS) ;
    ptr = new char [ Taille + 1] ;
    strcpy ( ptrch, ptrS)
  }
  Chaine :: ~ Chaine ( )
  { delete [ ] ptrch ;
    ptrch = NULL; }
  Chaine::
  unsigned Chaine :: GetTaille ( )
  { return Taille ; }
  unsigned Chaine :: Frequence_Caractère(char car)
  { unsigned Nbc = 0 ;
    if ( ptrch ! = NULL)
    { for (int i = 0 ; i < Taille ; i++ )
        if ( ptrch [i] == car )
          Nbc ++ ;
    }
    return Nbc ;
  }
  void Chaine :: Eliminer_Car ( Char car )
  { int j=0 ;
    if ( ptrch ! = NULL)
    { for ( int i = 0 ; i < Taille < i++ )
      { ptrch [j] = ptrch [i] ;
        if(ptrch [i] ! = car)
          j++ ,
      }
    ptrch[i] = '\0' ; Taille = j ; } }
```

CamScanner

```
void Chaine :: Afficher ( )
{ if (Taille != 0)
  { cout << "Cette chaine contient " << Taille <<
      "caracteres" << endl;
    cout << ptrch << endl;
  }
  else cout << "Cette chaine est vide \n";
}
```

Exercice 2 :

```
#include <iostream>
#include < cctype>
#include < cstdlib>
using namespace std;
// Classe hasard statique
Class CHasardST
{ int Val[10],
  Public :
    CHasardST ( int max) ;
    void Afficher ( ),
};
Class CHasardDY
{ int *ptrVal;
  short NbVal;
  Public :
  CHasardDY (int max, short Nb) ;
  ~CHasardDY( ) ;
  void Afficher( ),
};
int main ( )
{ // Hasard statique
  int Mx ;
  cout << " Préciser la valeur maximale de
       valeurs aléatoires à générer " ;
  cin >> Mx ;
  cout << " Valeurs aléatoires [0," << Mx <<
       " ] : \n" ;
CHasard SuiteS (Mx) ;
SuiteS. Afficher( ) ;
// Hasard dynamique
int Mx ;
short NbR ;
cout << " Préciser la valeur maximale des
  valeurs aléatoires à afficher:" ;
cin >> Mx ;
cout << " Leur Nombre: " ;
cin >> NbR ;
CHasardDY SuiteD (Mx, NbR) ;
SuiteD. Afficher( ) ;
return 0 ;
}
//Définition des méthodes
CHasardST :: CHasardST( int Mx)
{ for ( int i = 0 ; i < 10 ; i++ )
    val[ i] = ( rand( ) / RAND_MAX) * Mx ;
}
```

pour obtenir des val réelles — (double)

```
CHasardST :: Afficher( )
{
  for( int i=0 ; i<10 ; i++ )
    cout << val[i] << '\t' ;
  cout<< endl;
}
//Classe dynamique
CHasardDy :: CHasardDy( int Max, short NB)
{ NbVal = Nb;
  ptrVal = new int [NbVal] ;
  for( int i = 0 ; i < NbVal ; i++)
    ptrVal[i] = (rand() / RAND_MAX) * Max;
}
void CHasardDY::Afficher()
{ cout << NbVal << " valeurs aléatoires: \n";
  for( int i=0 ; i<NbVal ; i++ ;
    cout << ptrVal[i] << '\t';
  cout << endl;
}
CHasardDY :: ~ CHasardDY ( )
{ delete [] ptrVal;
  NbVal = 0; ptrVal= NULL;
}
```

CamScanner