# Série 3 (Corrigé)

## Exo1 :

a. La programmation est, ici, classique. Nous avons simplement défini une constante `NVAL` destinée à contenir le nombre de valeurs du tableau. Notez bien que la déclaration `int t[NVAL]` est acceptée puisque `NVAL` est une « expression constante ».

```
#include <iostream>
using namespace std ;
main()
{   const int NVAL = 10 ;              /* nombre de valeurs du tableau */
    int i, min, max ;
    int t[NVAL] ;
    cout << "donnez " << NVAL << " valeurs\n"  ;
    for (i=0 ; i<NVAL ; i++) cin >> t[i] ;
    max = min = t[0] ;
    for (i=1 ; i<NVAL ; i++)
      { if (t[i] > max) max = t[i] ;  /* ou max = t[i]>max ? t[i] : max */
        if (t[i] < min) min = t[i] ;  /* ou min = t[i]<min ? t[i] : min */
      }
    cout << "valeur max : " << max << "\n" ;
    cout << "valeur min : " << min << "\n" ;
}
```

b. On peut remplacer systématiquement `t[i]` par `*(t+i)`. Voici finalement le programme obtenu :

```
#include <iostream>
using namespace std ;
main()
{   const int NVAL = 10 ;              /* nombre de valeurs du tableau */
    int i, min, max ;
    int t[NVAL] ;
    cout << "donnez " << NVAL << " valeurs\n"  ;
    for (i=0 ; i<NVAL ; i++) cin >> *(t+i) ;
    max = min = *t ;
    for (i=1 ; i<NVAL ; i++)
      { if ( *(t+i) > max) max = *(t+i) ;
        if ( *(t+i) < min) min = *(t+i) ;
      }
    cout << "valeur max : " << max << "\n" ;
    cout << "valeur min : " << min << "\n" ;
}
```

## Exo2 :

En C++, par défaut, les arguments sont transmis par valeur. Mais, dans le cas d'un tableau, cette valeur, de type pointeur, n'est rien d'autre que l'adresse du tableau. Quant à la transmission par référence, elle n'a pas de signification dans ce cas. Nous n'avons donc aucun choix concernant le mode de transmission de notre tableau.
En ce qui concerne le nombre d'éléments du tableau, on peut indifféremment en transmettre l'adresse (sous forme d'un pointeur de type `int *`), ou la valeur ; ici, la seconde solution est la plus appropriée, puisque la fonction n'a pas besoin d'en modifier la valeur.
En revanche, en ce qui concerne le maximum et le minimum, ils ne peuvent pas être transmis par valeur, puisqu'ils doivent précisément être déterminés par la fonction. Il faut donc obligatoirement prévoir de passer:

- soit des références. L'en-tête de notre fonction se présentera ainsi :

```
void maxmin (int t[], int n, int & admax, int & admin)
```

- soit des pointeurs sur des `float`. L'en-tête de notre fonction se présentera ainsi :

```
void maxmin (int t[], int n, int * admax, int * admin)
```

L'algorithme de recherche de maximum et de minimum peut être calqué sur l'exercice précédent, en remplaçant `max` par `*admax` et `min` par `*admin`. Voici ce que pourrait être notre fonction :

- avec transmission par référence :

```
void maxmin (int t[], int n, int & admax, int & admin)
{   int i ;
    admax = t[0] ;
    admin = t[0] ;
    for (i=1 ; i<n ; i++)
       { if (t[i] > admax) admax = t[i] ;
         if (t[i] < admin) admin = t[i] ;
       }
}
```

- avec transmission par pointeurs

```
void maxmin (int t[], int n, int * admax, int * admin)
{   int i ;
    *admax = t[0] ;
    *admin = t[0] ;
    for (i=1 ; i<n ; i++)
       { if (t[i] > *admax) *admax = t[i] ;
         if (t[i] < *admin) *admin = t[i] ;
       }
}
```

Ici, si l'on souhaite éviter les « indirections » qui apparaissent systématiquement dans les instructions de comparaison, on peut travailler temporairement sur des variables locales à la fonction (nommées ici `max` et `min`). Cela nous conduit à une fonction de la forme suivante :

```
void maxmin (int t[], int n, int * admax, int * admin)
{  int i, max, min ;
   max = t[0] ;
   min = t[0] ;
   for (i=1 ; i<n ; i++)
      {  if (t[i] > max) max = t[i] ;
         if (t[i] < min) min = t[i] ;
      }
   *admax = max ;
   *admin = min ;
}
```

Voici un petit exemple de programme d'utilisation de la première fonction :

```
#include <iostream>
using namespace std ;
main()
{  void maxmin (int [], int, int &, int &) ;
   int t[8] = { 2, 5, 7, 2, 9, 3, 9, 4} ;
   int max, min ;
   maxmin (t, 8, max, min) ;
   cout << "valeur maxi : " << max << "\n" ;
   cout << "valeur mini : " << min << "\n" ;
}
```

Et voici le même exemple utilisant la seconde fonction :

```
#include <iostream>
using namespace std ;
main()
{  void maxmin (int [], int, int *, int *) ;
   int t[8] = { 2, 5, 7, 2, 9, 3, 9, 4} ;
   int max, min ;
   maxmin (t, 8, &max, &min) ;
   cout << "valeur maxi : " << max << "\n" ;
   cout << "valeur mini : " << min << "\n" ;
}
```

## Exo 3:

Par analogie avec ce que nous avions fait dans l'exercice 1, nous pourrions songer à déclarer le tableau concerné dans l'en-tête de la fonction sous la forme `t[][]`. Mais cela n'est plus possible car, cette fois, pour déterminer l'adresse d'un élément `t[i][j]` d'un tel tableau, le compilateur doit en connaître la deuxième dimension.

Une solution consiste à considérer qu'on reçoit un pointeur (de type `float*`) sur le début du tableau et d'en parcourir tous les éléments (au nombre de `n*p` si `n` et `p` désignent les dimensions du tableau) comme si l'on avait affaire à un tableau à une dimension.

Cela nous conduit à cette fonction :

```
float somme (float * adt, int n, int p)
{  int i ;
   float s ;
   for (i=0 ; i<n*p ; i++)  s += adt[i] ;     /* ou s += *(adt+i) */
   return s ;
}
```

Pour utiliser une telle fonction, la seule difficulté consiste à lui transmettre effectivement l'adresse de début du tableau, sous la forme d'un pointeur de type `int *`. Or, avec, par exemple `t[3][4]`, `t`, s'il correspond bien à la bonne adresse, est du type « pointeur sur des tableaux de 4 flottants ». A priori, toutefois, compte tenu de la présence du prototype, la conversion voulue sera mise en œuvre automatiquement par le compilateur.

Voici finalement un exemple d'un tel programme d'utilisation de notre fonction :

```
#include <iostream>
using namespace std ;
main()
{  float somme (float *, int, int) ;
   float t[3] [4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} } ;
   cout << "somme : " << somme ((float *)t, 3, 4) << "\n" ;}
```

## Exo 4 :

Il nous faut utiliser deux variables (`adt1` et `adt2`) de type pointeur sur des entiers pour conserver les adresses des emplacements alloués pour chacun des deux tableaux d'entiers.

```
#include <iostream>
using namespace std ;
main()
{ int nval ;            //  nombre de valeurs
  int * adt1, * adt2 ;  // attention, pas int * adt1, adt2
  do { cout << "combien de valeurs : " ;
       cin >> nval ;
     }
  while (nval <= 0) ;    // on refuse les valeurs négatives
      /* allocation premier tableau, lecture valeurs */
  adt1 = new int [nval] ;
  cout << "donnez " << nval << " valeurs \n ";
  for (int i = 0 ; i<nval ; i++) cin >> adt1[i] ;   // ou cin * (adt1+i)
     /* allocation second tableau, calcul des carrés */
  adt2 = new int [nval] ;
  for (int i = 0 ; i<nval ; i++) adt2[i] = adt1[i] * adt1[i] ;
     /* suppression premier tableau, affichage valeurs */
  delete adt1 ;
  cout << "voici leurs carrés : \n" ;
  for (int i = 0 ; i<nval ; i++) cout << adt2[i] << " " ;
     /* suppression du second tableau */
  delete adt2 ;
}
```

Notez que, dans l'appel de l'opérateur `delete`, il n'est pas nécessaire de préciser le nombre d'éléments du tableau d'entiers à libérer. Il n'en ira plus de même, lorsque l'on aura affaire à des tableaux d'objets.
Voici un exemple d'exécution de ce programme :

```
combien de valeurs : 0
combien de valeurs : -2
combien de valeurs : 8
donnez 8 valeurs
 1 2 5 9 7 3 0 8 6 -2
```

## Exo5 :

Une démarche consiste à créer un « tableau de 7 pointeurs sur des chaînes », correspondant chacune au nom d'un jour de la semaine. Comme ces chaînes sont ici constantes, il est possible de créer un tel tableau par une déclaration comportant une initialisation de la forme :

```
char * jour [7] = { "lundi", "mardi", ...
```

N'oubliez pas alors que `jour[0]` contiendra l'adresse de la première chaîne, c'est-à-dire l'adresse de la chaîne constante `"lundi"`; `jour[1]` contiendra l'adresse de `"mardi"`...

Pour afficher la valeur de la chaîne de rang `i`, il suffit de remarquer que son adresse est simplement `jour[i-1]`.
D'où le programme demandé :

```
#include <iostream>
using namespace std ;
main()
{
   char * jour [7] = { "lundi",    "mardi",  "mercredi", "jeudi",
                       "vendredi", "samedi", "dimanche"
                     } ;
   int i ;
   do
      { cout << "donnez un nombre entier entre 1 et 7 : " ;
        cin >> i ;
      }
   while ( i<=0 || i>7) ;
   cout << "le jour numéro " << i << " de la semaine est " << jour[i-1] ;}
```