Chapitre 8

Tableaux

8.1 Introduction

Les variables que nous avons utilisées jusqu'à maintenant ne permettent de stocker qu'une seule donnée. A chaque fois que l'on souhaitait stocker une donnée, on créait une variable. Par exemple, pour stocker le score d'un joueur on crée une variable `scoreJoueur` ; pour stocker le score de deux joueurs, on crée deux variables, `scoreJoueur1` et `scoreJoueur2`. Cette méthode est adaptée tant que le nombre de données est faible. Mais si l'on doit stocker le score de 1 000 joueurs, il n'est pas réaliste de vouloir créer 1 000 variables pour cela. Dans ce cas, on utilise une structure de données que l'on appelle tableau.

Un **tableau** est un ensemble de valeurs de même type (par exemple `int`) placées les unes à côté des autres dans la mémoire.

Déclarer un tableau revient à regrouper dans une même variable (un seul nom de variable) un nombre déterminé N d'éléments *de même type* plutôt que de déclarer N variables de noms différents. On peut considérer qu'à partir de 5, le nombre de données est *grand* et il faut penser à utiliser un tableau. Par exemple pour :

- les notes obtenues par l'étudiant Dupond en algorithmique pour l'année 2017,
- les températures relevées à midi tous les jours de l'année 2017,
- les codes des articles vendus dans un magasin.

8.2 Déclarer un tableau

On déclare une variable de type tableau en donnant d'abord le type de données qu'il va contenir, puis le nom de la variable, suivi entre crochets du nombre de cases du tableau.

```
1 type nomTableau[TAILLE];
```

où `TAILLE` est une valeur entière *constante*. Par exemple, l'instruction

```
1 int tab[1000];
```

réserve en mémoire un tableau de 1000 cases pouvant contenir des valeurs de type `int` et lui associe le nom de variable `tab`.

Remarque : comme pour les autres types de variables, une variable de type tableau est détruite dès que le programme rencontre l'accolade fermante correspondant au bloc d'instructions où la variable est déclarée.

8.2.1 Distinction taille / nombre d'éléments

La taille (ou capacité) du tableau doit être une *constante* et non une variable. Or, le plus souvent, on ne connaît pas le nombre de valeurs qu'on devra stocker dans le tableau. Il faut alors déclarer un tableau de taille supérieure aux besoins du programme, et utiliser une variable pour suivre le nombre d'éléments (i.e. de valeurs stockées) durant le programme.

Il faut bien faire la différence entre la taille du tableau (donnée entre crochets à la déclaration), le nombre de valeurs qu'on prévoit de stocker lors d'une exécution et le nombre de valeurs effectivement stockées.

8.2.2 Constantes

Dans l'exemple précédent, la taille a été donnée par une constante littérale : `1000`. La bonne pratique veut qu'on utilise des constantes nommées plutôt que des nombres arbitraires. On utilise pour cela une *macro* de la forme suivante à placer en en-tête du programme (avant les fonctions et le `main()`) :

```
1 #define NOM_CONSTANTE valeur
```

Une constante ainsi définie est accessible n'importe où dans le programme : sa portée n'est pas limitée. Il ne s'agit pas d'une variable. Le préprocesseur remplace la macro (le nom de la constante) par sa valeur partout dans le code avant la compilation. Pour les distinguer des variables, les noms des constantes sont en MAJUSCULES. Il est d'usage de définir la taille d'un tableau avec une constante nommée `TAILLE` ou `TAILLE_MAX` par exemple.

Comme pour les autres types de variables, déclarer un tableau ne suffit pas à l'initialiser, c'est-à-dire remplir les cases du tableau. Voyons donc comment accéder au contenu.

8.3 Déclaration et initialisation conjointes

Il est possible de déclarer et initialiser un tableau en une ligne, sans boucle. Pour cela, il faut donner la valeur des éléments entre accolades. Ainsi, l'instruction

```
1 int tab[3] = {1, 3, 5};
```

crée un tableau de 3 éléments dont le premier vaut 1, le deuxième 3 et le troisième 5. Si la taille du tableau est supérieure au nombre de valeurs fournies, celles-ci remplissent les premières cases. Dans le cas d'un tableau d'entiers, les autres éléments sont initialisés à 0. Ainsi,

```
1 int tab[6] = {2, 3};
```

a le même effet que

```
1 int tab[6] = {2, 3, 0, 0, 0, 0};
```

Au moins une valeur doit être donnée entre accolades.

On peut aussi utiliser cette syntaxe sans fournir la taille entre crochets. Dans ce cas, la taille est prise égale au nombre de valeurs fournies entre accolades. Ainsi,

```
1 int tab[] = {18, 32, 23, 1, 0};
```

déclare un tableau de taille 5, dont les éléments sont initialisés avec les valeurs fournies.

Attention : Il n'est pas possible d'initialiser un tableau avec un autre tableau. Ainsi, le code suivant est incorrect :

```
1 int tab[] = {1, 2, 3};
2 int t2[] = tab; // impossible
```

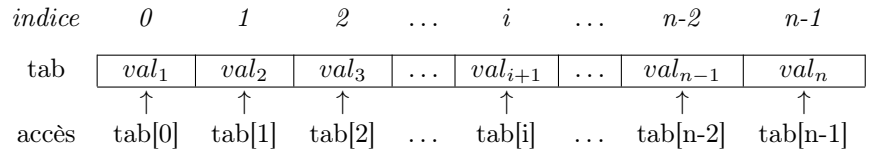
▶ Sur ce thème : **Exercice 7 du TP 8**

8.4 Accéder aux éléments

On accède aux éléments d'un tableau par leur position dans le tableau. C'est un entier que l'on appelle *indice* de l'élément. On donne le nom du tableau suivi de l'indice entre crochets, comme ceci :

```
1 nomTableau[indice];
```

Attention : L'indice d'un tableau de n éléments commence à 0 et finit à $n-1$.



Chaque élément d'un tableau peut être vu comme une variable (c'est une zone mémoire dans laquelle on stocke une donnée). On peut donc affecter une valeur à un élément du tableau ou se servir de la valeur stockée pour un calcul ou un affichage. Par exemple

```
1  int main()
2  {
3    // supposons que les trois premières cases contiennent 45, 5 et 1
4
5    affichage("le premier élément du tableau tab est : ", tab[0], '\n');
6    affichage("le deuxième élément du tableau tab est : ", tab[1], '\n');
7    affichage("le troisième élément du tableau tab est : ", tab[2], '\n');
8
9    tab[0] = 25;
10   tab[0] += tab[1];
11
12   affichage("le premier élément du tableau tab est maintenant : ", tab[0], '\n');
13   return 0;
14 }
```

affichera :

```
le premier élément du tableau tab est : 45
le deuxième élément du tableau tab est : 5
le troisième élément du tableau tab est : 1
le premier élément du tableau tab est maintenant : 30
```

Attention : Une variable de type tableau ne peut être l'objet d'une affectation sans être indicée (c'est-à-dire suivie d'un indice entre crochets), sauf dans le cas d'une déclaration-initialisation conjointe (voir plus loin).

▶ Sur ce thème : **Exercice 1 du TD 8**

8.5 Parcourir un tableau

Parcourir un tableau, c'est-à-dire atteindre successivement tous ses éléments (dans l'ordre croissant ou décroissant des indices), implique nécessairement une boucle, dont le nombre d'itérations est le nombre d'éléments du tableau.

Par exemple, pour initialiser un tableau de taille 5 avec les premiers éléments de la table de 7 :

```
1 int tab[5]; // déclaration du tableau
2 int i = 0;  // indice
3
```

```
while (i<5) { // l'indice parcourt les éléments
    tab[i] = 7*i;
    affichage(tab[i], '\n');
    i++;
}
```

L'affichage sera

```
0
7
14
21
28
```

Pour ajouter 2 aux éléments et afficher dans l'ordre décroissant des indices,

```
int i=4;
while (i>=0) {
  tab[i] = tab[i] + 2; // ajoute 2 à l'élément d'indice i
  affichage(tab[i], '\n');
  i--;
}
```

L'affichage sera

```
30
23
16
9
2
```

► Sur ce thème : **Exercices 2, 5 et 6 du TD 8**

```
int main ()
{
  // déclaration d'un tableau pouvant contenir au plus 1000 entiers
  int tab[1000];

  // saisie du nombre de valeurs à stocker
  int nbElem = 0;
  while (nbElem < 1 || nbElem > 1000) {
    affichage("Nombre de valeurs à stocker ? (entre 1 et 1000) : ");
    saisie(nbElem);
  }

  // saisie des valeurs
  int i = 0;
  while (i < nbElem) {
    saisie(tab[i]);
    i++;
  }

  // le nombre de valeurs stockées est bien nbElem
  affichage("nombre d'éléments dans le tableau : ", nbElem, '\n');

  // et les indices des éléments sont compris entre 0 et nbElem-1
  while (i < nbElem) {
    affichage("élément numéro", i, "du tableau : ", tab[i], '\n');
    i++;
  }
```

```
28   return 0;
29 }
```

1000 est la taille du tableau, soit le nombre de cases reservées à la déclaration ligne 4. Le nombre d'éléments sera forcément inférieur à cette taille. Il aurait été préférable de nommer la constante par la macro `#define TAILLE_MAX 1000` et remplacer tous les `1000` par des `TAILLE_MAX` ;

nbElem est le nombre d'éléments à placer dans ce tableau de 1000 cases, fourni par l'utilisateur. C'est aussi le nombre d'itérations à chaque parcours du tableau.

▶ Sur ce thème : **Exercices 5 du TD8 et tous les autres aussi !**

TD8 : Tableaux (Corrigé)

✓ Exercice 1 : Test de compréhension *

Question 1.1 : [Indice] Qu'est-ce que l'indice d'un élément dans un tableau ? Quelle(s) valeur(s) peut-il prendre ?

Correction :

L'indice d'un élément correspond à la position de cet élément dans le tableau. L'indice est un nombre compris entre 0 et `N-1` pour un tableau comprenant `N` éléments. ◇

Question 1.2 : [Indice et valeur] Quelle différence y a-t-il entre l'indice d'un élément d'un tableau et sa valeur ?

Correction :

Ce sont deux choses différentes. L'indice est un entier correspondant à la position de l'élément dans un tableau alors que la valeur est une donnée (nombre, chaîne de caractères, etc) stockée dans le tableau. ◇

✓ Exercice 2 : Parcours *

Question 2.1 : Déclarer un tableau de nombres réels de taille 6 qui sera initialisé avec des valeurs quelconques.

Question 2.2 : Afficher le tableau à l'endroit puis à l'envers.

Correction :

```
int main ()
{
  double tab[] = {1.2, 2.3, 4.3, -6.6, 9., -23.3};

  affichage("Affichage du tableau à l'endroit :\n");
  int i =0 ;
  while (i < 6)
  {
    affichage(tab[i], '\n');
    i++ ;
  }

  affichage("Affichage du tableau à l'envers :\n");
  i=5;
  while ( i >=0)
  {
    affichage(tab[i], '\n');
    i-- ;
  }
  return 0;
}
```

◇

✓ Exercice 3 : Différence *

Écrire un programme qui calcule la différence des éléments de rang pair et impair dans un tableau d'entiers. Par exemple si le tableau `32, 5, 12, 83, 3, 75, 2, 15` la différence sera égale à $32 - 5 + 12 - 83 + 3 - 75 + 2 - 15 = -129$.

Correction :

```c
#define TAILLE 8

int main ()
{
        int t[TAILLE] = {32, 5, 12, 83, 3, 75, 2, 15};
        int somme = 0;
        int i = 0;
        while (i < TAILLE) {
                if (i%2)
                   somme -= t[i];
                else
                   somme += t[i];
                i++;
        }
        return 0;
}
```

✓ Exercice 4 : Vérification de tri **

Écrire un programme qui vérifie si un tableau de 10 entiers fourni par l'utilisateur est trié dans l'ordre croissant ou non.

Correction :

```c
int main(void)
{
  int tab[10];
  int i=0;

  while(i<9)
 {
  affichage(i+1,"ème élément du tableau ?");
  saisie(tab[i]);
  i++;
 }

  i=0;
  while (i < 9 && (tab[i] > tab[i+1]))
      i++;

  if (i!=9)
    affichage("le tableau n'est pas trié \n");
  else
    affichage("le tableau est trié \n");

  return 0;
}
```

® Exercice 5 : Recherche du maximum *

Écrire un programme qui recherche le plus grand élément présent dans un tableau donné. Par exemple, si le tableau est égal à `{32, 5, 12, 8, 3, 75, 2, 15}`, ce programme doit afficher : `le plus grand élément du tableau a la valeur 75`.
Vous ferez attention à ne pas utiliser de constante explicite pour la taille du tableau !

Correction :

```c
#define TAILLE 8

int main ()
{
  int t[TAILLE] = {32, 5, 12, 83, 3, 75, 2, 15};
  int i = 1;
  int plusgrand = t[0];

  while (i< TAILLE)
  {
    if (t[i] > plusgrand)
      plusgrand = t[i] ;
    i ++;
  }

  affichage("le plus grand element du tableau a la valeur ", plusgrand, '\n');
  return 0;
}
```

Ⓡ Exercice 6 : Plus grand nombre pair **

Question 6.1 : Écrire un programme affichant l'entier pair de plus petit indice contenu dans un tableau d'entiers ainsi que l'indice de cet entier. Attention, si le tableau ne contient aucun entier pair, le programme doit afficher le message `Le tableau ne contient aucun entier pair`.

Question 6.2 : Compléter le programme précédent pour qu'il affiche le plus grand entier pair contenu dans un tableau d'entiers.

Correction :

```c
#define TAILLE_MAX 10
int main ()
{
  int tab[TAILLE_MAX] = {-5, 8 , 98, 45, -89, -102, 56, 1111, -79, 51};
  int i=0;

  //Affichage du tableau
  affichage("Affichage du tableau :\n");

  while (i < TAILLE_MAX)
  {
    affichage(tab[i], '\n');
    i++;
  }

  // recherche de l'entier pair de plus petit indice
  i = 0;
  int IndplusGrandEntierPair = -1;
  while (i < TAILLE_MAX && (IndplusGrandEntierPair==-1))
  {
    if (tab[i]%2==0
      IndplusGrandEntierPair = i;
    i++;
  }
  if (IndplusGrandEntierPair !=-1)
    affichage("L'entier pair de plus petit indice est : ", tab[IndplusGrandEntierPair],
              "\n Il correspond à l'indice", IndplusGrandEntierPair, "\n");
  else
```

```
29      affichage("Le tableau ne contient pas de nombre pair \n");
30
31    // recherche eventuelle du plus grand entier par
32     if (IndplusGrandEntierPair !=-1)
33     {
34       i= IndplusGrandEntierPair +1;
35       while (i < TAILLE_MAX)
36       {
37         if (tab[i]%==0 && tab[i]>tab[IndplusGrandEntierPair])
38            IndplusGrandEntierPair = i;
39         i++;
40       }
41     affichage("Le plus grand entier pair est : ", tab[IndplusGrandEntierPair], '\n');
42     }
43     return 0;
44   }
```

TP8 : Tableaux (Corrigé)

✓ Exercice 7 : Les valeurs par défaut *

Écrire un programme permettant d'identifier les valeurs prises par défaut lors de l'initialisation d'un tableau avec une liste de littéraux trop courte pour les tableaux de type `int`, `double`, `char`, `string` et `bool`.

Correction :

```
int main()
{
  int tabint[2] = {1} ;
  affichage("-", tabint[1], "-\n");

  double tabdouble[2] = {1.0} ;
  affichage("-", tabdouble[1], "-\n");

  char tabchar[2] = {'a'} ;
  affichage("-", tabchar[1], "-\n");

  string tabstring[2] = {"a"} ;
  affichage("-", tabstring[1], "-\n");

  bool tabbool[2] = {true} ;
  affichage("-", tabbool[1], "-\n");

  return 0;
}
```

✓ Exercice 8 : Moyennes des nombres positifs d'un tableau*

Écrire un programme qui initialise un tableau de 15 entiers tirés aléatoirement entre -10 et 10. Le programme affiche ensuite la moyenne des nombres positifs de ce tableau.

Correction :

```
#define TAILLE 15

int main()
{
  srand(time(NULL));
  int tab[TAILLE];
  int i = 0;
  while (i < TAILLE)
  {
    tab[i] = rand() % 21 - 10;
    i++;
  }

  int nbPos = 0;
  int sommePos = 0;

  i = 0 ;
  while (i < TAILLE)
  {
```

```
20      if (tab[i] > 0)
21      {
22        sommePos += tab[i];
23        nbPos++;
24      }
25      i++;
26    }
27
28    if (nbPos > 0)
29      affichage("Moyenne des positifs : ", (double) sommePos / nbPos, '\n' );
30    else
31      affichage("Aucun nombre positif\n");
32
33    return 0;
34  }
```

✓ Exercice 9 : Rotations **

Écrire un programme qui permet d'effectuer une rotation à droite des éléments d'un tableau d'entiers, quelque soit le nombre d'éléments qu'il contient.

Exemple d'affichage attendu :

```
Avant rotation a droite
  {15, -10, 70, 74, 12, 20}
Apres rotation a droite
  {20, 15, -10, 70, 74, 12}
```

Correction :
On peut passer rapidement sur l'affichage !

```
1   #define TAILLE_MAX 6
2
3   int main()
4   {
5     int tab[TAILLE_MAX] = {15,-10,70,74,12,20};
6
7     // Affichage
8     affichage("Avant rotation a droite \n");
9     int i=0;
10    while ( i < TAILLE_MAX)
11    {
12      affichage(tab[i], ", ");
13      i++ ;
14    }
15    affichage('\n');
16
17    // Rotation
18    int aux=tab[TAILLE_MAX-1];
19    i = TAILLE_MAX - 1;
20    while (i>=1)
21    {
22      tab[i]=tab[i-1];
23      i--;
24    }
25    tab[0]=aux;
26
```

```
27    // Affichage
28    affichage("Apres rotation a droite \n");
29    i=0;
30    while ( i < TAILLE_MAX)
31    {
32      affichage(tab[i], ", ");
33      i++ ;
34    }
35    affichage('\n');
36    return 0;
37  }
```

Exercice 10 : Affichage trié ***

Question 10.1 : Écrire un programme qui pour un tableau d'entiers donnés affiche le plus petit élément supérieur à une borne donnée `borne`. On supposera que les éléments sont distincts et compris entre 0 et 100.

Question 10.2 : Transformer ce programme de manière à afficher les éléments d'un tableau du plus petit au plus grand en prenant comme bornes successives 0, puis le plus petit élément du tableau, puis le second plus petit, *etc.*

Correction :

```
1   #define TAILLE 10
2
3   int main()
4   {
5     srand(time(NULL));
6     int tab[TAILLE] = {7, 48, 3, 2, 59, 17, 34, 10, 0, 22} ;
7     int borne = -1; // valeur minimale à surpasser
8     int nb_parcours = 0; // nb de parcours ud tableau
9
10    int i = 0;
11    while (nb_parcours < TAILLE)
12    {
13      int min = 101;
14      i = 0;
15      while (i < TAILLE)
16      {
17        if (tab[i] > borne && tab[i] < min)
18          min = tab[i];
19        i++;
20      }
21
22      affichage(min, '\n');
23      borne = min;
24      parcours++;
25    }
26    return 0;
27
28  }
```