# TD 1 – Tableaux

## Vecteurs

### 1) Initialisation d'un tableau

a) Écrire une fonction prenant en argument un tableau de type `array[1..N] of integer`, et le remplissant de zéros.

b) Écrire une fonction prenant en argument un tableau `t` de type `array[1..N] of integer`, et le remplissant avec les valeurs d'une fonction `f` déclarée ailleurs dans le code. La case `t[i]` doit contenir la valeur `f(i)`.

c) Écrire une fonction prenant en argument un tableau de type `array[1..N] of integer`, et le remplissant de valeurs aléatoires.

*Indication :* on rappelle que `random(m)` renvoie un entier pseudo-aléatoire compris entre 0 et $m$.

### 2) Impression d'un tableau

a) Écrire une fonction prenant en argument un tableau de type `array[1..N] of integer`, et l'affichant sur la console. Le tableau devra apparaître sous la forme

```
[ 1; 2; 5; 3; 7; 18; 1]
```

b) Modifier cette fonction de manière à ce qu'elle n'imprime que les 20 premiers éléments du tableau, suivis de points de suspension si `N` est plus grand que 20.

### 3) Moyennes

a) Écrire une fonction `moyenne` prenant en argument un tableau de réels, et renvoyant la moyenne de ses éléments.

b) Écrire une fonction `moyenne_elaguee` prenant en argument un tableau de réels, et renvoyant de même la moyenne de ses éléments, *élaguée du plus petit et du plus grand élément du tableau.*

Par exemple, la moyenne élaguée du tableau `[12.0; 14.5; 8.1; 15.3]` est `13.25=(12+14.5)/2`, car `8.1` et `15.3` sont enlevés de la série.

### 4) Appliquer une fonction aux éléments d'un tableau

a) Écrire une fonction prenant en argument un tableau d'entier `t` de type `array[1..N] of integer`, et augmentant tous ses éléments de 1.

b) Modifier votre fonction de manière à ce qu'elle remplace chaque élément de `t` par son image par la fonction `f` définie autre part dans le code.

### 5) Ce tableau est-il trié ?

a) Écrire une fonction `croissant` prenant en argument un tableau de type `array[1..N] of integer`, et renvoyant vrai si et seulement si le tableau est trié en ordre croissant.

b) Modifier cette fonction de manière à ce qu'elle renvoie 0 si le tableau n'est pas trié, 1 si le tableau est trié en ordre croissant, et $-1$ s'il est trié en ordre décroissant.

Attention au fait que les éléments du tableau ne sont pas nécessairement tous distincts.

### 6) Miroir d'un tableau

a) Écrire une fonction `miroir` prenant en argument un tableau de type `array[1..N] of integer`, et en renvoyant l'*image miroir*, en échangeant `t[1]` et `t[N]`, `t[2]` avec `t[N-1]`, etc.

b) Modifier votre fonction de manière à ce qu'elle prenne deux arguments supplémentaires `u` et `v`, et qu'elle retourne le tableau `t` après avoir retourné la partie constituée des case d'indices compris entre `u` et `v` (inclus). Cette fonction sera utilisée plus bas.

### 7) Avant de trier, on mélange, et on se fait un petit poker !

On souhaite réaliser une procédure qui mélange aléatoirement les éléments d'un tableau donné en paramètre (par exemple pour simuler le mélange d'un jeu de carte).

a) Écrire une procédure prenant en argument un tableau d'entiers de type `array[1..N] of integer` et deux entiers `i` et `j`, et échangeant les éléments d'indices `i` et `j` du tableau.

En déduire une première procédure de mélange.

b) On souhaite simuler la méthode de mélange qu'on voit souvent dans les films : on coupe le jeu en deux, et on insère les cartes d'un des deux paquets entre les cartes de l'autre.

Pour cela, on va composer un nouveau tableau, et le remplir en prenant aléatoirement des éléments de la première ou de la seconde partie du tableau initial. Écrire la procédure mettant en oeuvre cette idée.

c) Utiliser ces deux méthodes de mélange pour simuler un grand nombre de distribution de cinq cartes d'un jeu de 52 cartes, et calculant la fréquence des figures usuelles du poker : une paire, deux paires, une couleur...

### 8) Recherche dans un tableau

a) Écrire une fonction prenant en argument un tableau d'entiers `t`, de type `array[1..N] of integer`, et un entier `x`, et renvoyant le nombre d'occurrences de `x` dans `t`.

b) Écrire une fonction prenant les mêmes arguments, et renvoyant 0 si `x` ne se trouve pas dans `t`, et l'indice de la première position (on dit *occurrence*) de `x` dans `t` dans le cas contraire.

### 9) Crible d'Erathostene

## Matrices

### 1) Initialisation d'un tableau à deux dimensions

a) Écrire une fonction prenant en argument une matrice de type `array[1..N,1..M] of integer`, et initialisant l'ensemble de ses cases à 0.

b) Modifier cette fonction pour qu'elle remplisse le tableau d'entiers positifs aléatoires.

### 2) Moyenne des éléments d'une matrice

Écrire une fonction calculant la moyenne des termes d'une matrice de type `array[1..N,1..M] of real`.

### 3)

a)

## Des exercices plus compliqués

### 1) Rotation d'un tableau

On veut écrire une fonction `rotation` prenant en argument un tableau de type `array[1..N] of integer`, et un entier $k$, et renvoyant la *rotation* d'ordre $k$ de ce tableau : `t[1]` va en `t[k+1]`, `t[2]` et `t[k+2]`,... `t[N-k-1]` en `t[N]`, `t[N-k]` en `t[1]`,... et `t[N]` en `t[k]`.

a) Une première idée consiste à utiliser un tableau auxiliaire pour recopier le premier tableau, avant de faire une boucle pour remplir le tableau à modifier. Implémenter cette idée.

b) Une deuxième idée, plus difficile à mettre en oeuvre, consiste à sauvegarder `t[1]`, placer `t[k+1]` dans `t[1]`, puis `t[2k+1]` dans `t[k+1]`, et ainsi de suite.

Si tout se passe bien (quand est-ce le cas ?), l'algorithme se termine en une seule passe. Dans le cas contraire, il faut poursuivre par la première case qui n'a pas encore été modifiée, et ainsi de suite.

Implémenter cette idée.

c) Une idée beaucoup plus simple, mais plus astucieuse, consiste à constater qu'on peut se ramener à des images miroirs. En effet, si la structure du tableau est | A | B |, on peut obtenir le résultat final | B | A | en prenant le miroir de la partie A, puis le miroir de la partie B, et enfin le miroir du tableau obtenu. Faire un petit dessin pour vérifier ceci, et implémenter cette idée à l'aide de la fonction `miroir` construite plus haut.

### 2) Recherche dans un tableau, version ultra-rapide

Lorsque le tableau dans lequel on effectue une recherche est trié (en ordre croissant), on peut accélérer grandement la recherche en utilisant la méthode de *dichotomie*. Elle s'appuie sur le pseudo-algorithme suivant :

- au départ, l'élément `x` cherché doit se trouver dans le tableau `t[1..N]` ; on initialise `u` à `1` et `v` à `N` ;
- (B) si `u>v`, alors `x` ne se trouve pas dans le tableau ; on retourne `0` ou `false`, selon ce qui nous intéresse ;

  sinon, soit `m=(u+v)/2` ; on compare `x` à `t[m]` :
  - si `x=t[m]`, on retourne `true`, ou la valeur de `m`
  - si `x<t[m]`, `x` ne peut se trouver que dans la partie `t[u,m-1]` du tableau ; on pose `v=m-1`, et on retourne en (B) ;
  - sinon (i.e. si `x>t[m]`), `x` ne peut se trouver que dans la partie `t[m+1,v]` du tableau ; on pose `u=m+1`, et on retourne en (B).

a)