TP – LES ALGORITHMES SUR TABLEAUX

Les différentes opérations algorithmiques sur un tableau sont :

- Parcours : traitement systématique sur toutes les cases du tableau
- Recherche : on cherche un élément précis dans un tableau
- Insertion et suppression
- Tri

La suite de ce ¨Cours/TP présente une façon classique de programmer les algorithmes de PARCOURS et RECHERCHE dans une approche guidée.

L'objectif à terme est d'être capable de reconnaître dans un énoncé le type d'algorithme à appliquer.

LE PARCOURS

Définition : nous parlons de parcours de tableau, lorsqu'on accède à tous les éléments d'un tableau en appliquant un même **traitement** sur chacun des éléments. Le **traitement** appliqué lors du parcours d'un tableau dépend du contexte et de l'objectif du programme. Ce traitement peut-être un affichage des éléments d'un tableau, une modification de valeur, un calcul, etc…

Un parcours utilise l'algorithme général suivant :

```
Pour i allant de 0 à (taille_du_tableau - 1) faire
        // appliquer le traitement sur tableau[i]
FinPour
```

La démarche proposée dans ce TP est identique au TP sur les fonctions :

1. Écrire la signature et le corps de la fonction
2. Tester la fonction dans un programme de test le plus simple possible.

LE PARCOURS EN LECTURE SEUL

Nous allons commencer ce TP avec des accès en **lecture seul** aux éléments du tableau.

À faire : nous allons coder un programme permettant l'affichage de tous les éléments d'un tableau. Nous allons utiliser une fonction dont voici la signature et le corps de la fonction. Copier-coller ce code dans votre projet.

```
/**
*
* @param A : le tableau d'entier à afficher. On remarque ici le formalisme : int A[]
* return value : void  ➔ cette fonction affiche seulement le tableau
*/
public static void afficheTableau(int A[]) {
    for (int i = 0; i < A.length; i++) {
        System.out.println(A[i] + " ");
    }
}
```

À faire : coder un programme de test permettant de valider le fonctionnement de la fonction « afficherTableau ». Copier-coller le code dans votre projet. Vérifier le bon fonctionnement.

```
public static void testAfficheTableau() {
    int  tab[]= {10, 20, 30, 40, 50} ;
    afficheTableau(tab);  //tab est passé en paramètre de afficheTableau
}
```

À faire : reprendre la fonction « afficheTableau » en ajoutant un paramètre « ordre » permettant de savoir si l'affichage doit se faire du premier élément au dernier, ou inversement.

```
public static void testAfficheTableau() {
    int  tab[]= {10, 20, 30, 40, 50} ;
    afficheTableau(tab,false);  //affiche 50,40,30,20,10
    afficheTableau(tab,true);  //affiche 10,20,30,40,50
}
```

Une autre syntaxe existe pour le parcours d'un tableau : « Pour chaque » ou « for each » en anglais. Attention, ce type de boucle ne permet pas le parcours inverse.

Dans le code suivant, « element » prend successivement la valeur de chaque valeur du tableau A. Son type (int) doit correspondre au type des éléments du tableau.

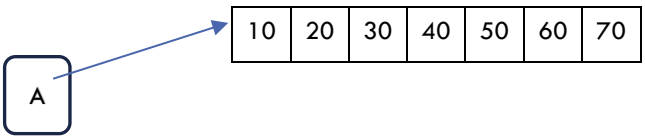
```
for (int element : A) {
     System.out.println("" + element);
}
// Cela peut se lire ainsi : « Pour chaque élément  du tableau A affiche-le ».
```

À faire : écrire une fonction « afficheTab_V2 » utilisant la version **for each**, ainsi qu'une fonction de test

LE PARCOURS EN ÉCRITURE

En java un tableau est un **objet**. Cela à une incidence dans la façon de l'utiliser dans une fonction. Quand celui-ci est passé en paramètre de fonction celui est passé **par référence.** Une **référence** C'est une sorte de "pointeur" vers l'emplacement mémoire où l'objet est stocké.

```
int A[] = {10,20,30,40,50,60,70} ;
```



Son fonctionnement diffère du passage de paramètre **par valeur** utilisé lors du TP sur les fonctions.

À faire : pour bien comprendre le point précédent, coder les fonctions suivantes :

```
public static void testAddUn() {
    int nb = 10;
    addUn(nb);
    System.out.println("dans la fonction de test nb vaut :" + nb);
}

public static void addUn(int a) {
    a=a+1;        //tab
    System.out.println("dans la fonction a vaut :" + a);
}
```

Dans la fonction « addUn » le paramètre « a » est un type natif (byte, short, int, long, boolean, float, double, char). Le paramètre « a » n'est pas un objet, c'est donc le **passage de paramètre par valeur** qui s'applique. Cela signifie qu'une **copie** de la valeur de la variable est transmise à la fonction. Toute modification de cette copie à l'intérieur de la fonction n'affectera pas la variable d'origine.

À faire : tester le programme précédent. Vous obtenez le résultat suivant :

```
dans la fonction a vaut :11
dans la fonction de test nb vaut :10
```

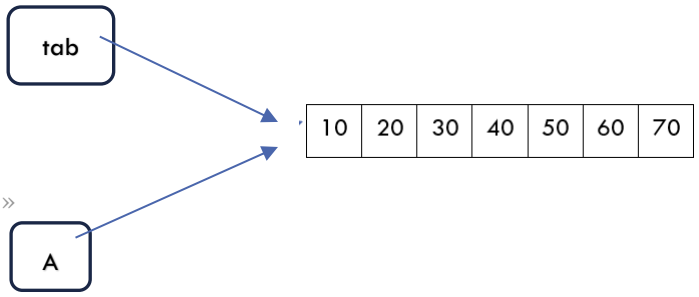
Vous remarquez que la modification de « a » dans la fonction « addUn » n'a pas été répercutée sur « nb » de « testAddUn ». C'est le comportement normal d'un passage de paramètre par valeur.

À faire : écrire une fonction « addUnTab » permettant d'ajouter +1 à tous les éléments du tableau. Copier-coller le code

```java
public static void testAddUnTab(){
   int tab[]= {10, 20, 30, 40, 50, 60, 70} ;
   addUnTab(tab);
   afficheTableau(tab, true);
   //affiche 11,21,31,41,51,61,71
}

// « A » et « tab » pointe au même endroit.
// En modifiant « A », on modifie aussi « tab »


public static void addUnTab(int A[]) {
    for (int i = 0; i < A.length; i++) {
       À[i] = A[i] +1 ;     //ajout de +1 a la case (i)
    }
}
```



Cet exemple permet de comprendre la différence entre passages de paramètre par **valeur** et par **référence**. Dans l'exemple « addUnTab » c'est la **référence** (et non une copie) du tableau qui est transmise. Cela signifie que les modifications apportées au tableau à l'intérieur de la fonction affectent le tableau original. Ce n'est pas le cas d'un passage par copie.

Conclusion : en Java, un tableau est un objet, donc il est **passé par référence** dans une fonction. Les modifications dans la fonction affectent directement le tableau original.

EXERCICE SIMPLE PARCOURS

Écrire et tester les fonctions suivantes pour un **tableau d'entier** :

- Écrire une fonction permettant de saisir « à la main » chaque case d'un tableau d'entier. Les valeurs autorisées seront comprises dans un intervalle « min max » autorisé.

```
Exemple : saisirTab(A,100,200) ;     // saisie des valeurs pour le tableau A.
                                     // les valeurs autorisées sont comprises entre 100 et 200
```

- Écrire une fonction permettant de compter le nombre de valeurs du tableau supérieur à une valeur passée en paramètre de fonction. Faire de même pour compter le nombre de valeurs du tableau inférieur à une valeur.

```
Exemple :      int v = compteValSup(A,10) ; // compte le nombre de valeurs de A supérieur à 10
               int v = compteValInf(A,10) ; // compte le nombre de valeurs de A inférieur à 10
```

- Écrire une fonction permettant de faire la somme de tous les éléments du tableau.

EXERCICE D'APPLICATION

Écrire une application pour « prof ». Cette application doit permettre au « prof » d'indiquer le nombre d'élèves de la classe. En fonction de ce nombre d'élèves, il saisit ensuite les notes pour un contrôle. Chaque note doit être entre 0 et 20. Il calcule ensuite la moyenne arrondie à l'entier le plus proche pour ce contrôle. Pour finir, il affiche le nombre d'étudiants ayant une note supérieure à la moyenne, inférieur à la moyenne, et égal à la moyenne.

EXERCICE DOUBLE BOUCLE

EXO1

Soit le tableau T1 suivant de N1 entiers :

3	6

Soit le tableau T2 suivant de N2 entier :

4	8	7	12

Écrire une fonction qui calcule l'opération suivante à partir des deux tableaux : il faut multiplier chaque élément du tableau 1 par chaque élément du tableau 2, et additionner le tout.

Cela donne :(3 * 4) + (3 * 8) + (3 * 7) + (3 * 12) + (6 * 4) + (6 * 8) + (6 * 7) + (6 * 12) = 279

EXO2

Soit un tableau A de N entiers. Chaque case contient le pourcentage de bonnes réponses aux N questions. Écrire une fonction permettant l'affichage sous forme de barre-graphe de ce tableau.

Exemple avec le tableau : int A[] = {52,32,15,19,80};

```
Question 1 **************************************************52%
Question 2 ********************************32%
Question 3 ***************15%
Question 4 ******************19%
Question 5 ********************************************************************************80%
```

FONCTION ET RETOUR DE TABLEAU

- Dans certaines situations, il peut être utile qu'une fonction retourne un tableau. Dans ce cas il faut indiquer le type de données du tableau dans la signature de la fonction (ex. int[], double[], etc.).
- Créer un tableau dans la fonction et le retourner celui-ci avec l'instruction return.

Exemple : on souhaite une fonction permettant de créer un tableau de N éléments remplis avec une valeur initiale passée en paramètre. La signature de la fonction « creerTab » est :

```
/**
*
* @param n : nombre de cases du tableau
* @param v : Valeur de remplissage
* @return int[]  :retourne un tableau de n cases remplit avec la valeur v
*/
public static int[] creerTab(int n, int v) {
        // à compléter
}
```

À faire : compléter la fonction « creerTab »

EXERCICES

- Écrire un programme permettant de faire la somme de 2 tableaux A et B dans un tableau C lui-même de N entiers.

Exemple : tableau A :

10	20	30	40	50	60	70

tableau B :

2	4	6	8	10	12	14

On obtient le tableau C :

12	24	36	58	60	72	84

- Écrire une fonction permet de récupérer un (sous)tableau contenant les valeurs négatives d'un tableau passé en paramètre.
- Écrire une fonction permettant de fusionner les éléments d'un tableau A et d'un tableau B dans un tableau C. Les tableaux A et B peuvent être de dimension différente. La dimension du tableau C est celle de A + celle de B

LA RECHERCHE

Nous parlons de **recherche** dans un tableau, lorsqu'on cherche un élément vérifiant une certaine propriété.

Cette recherche s'effectue jusqu'à la fin du tableau si l'élément cherché n'existe pas. Une recherche donne deux solutions possibles :

- L'élément cherché est trouvé ➔ retour de la valeur **true**, ou de la position où se trouve l'élément cherché
- L'élément cherché n'est pas trouvé ➔ retour de la valeur **false**, ou -1

Une recherche est différente d'un parcours puisqu'aucun traitement systématique n'est effectué sur tous les éléments. On peut quitter avant la fin du tableau, si l'élément est trouvé, sinon on jusqu'à la fin du tableau.

L'algorithme type est le suivant :

```
Pour i allant de 0 à taille du tableau - 1 Faire
   Si tableau[i] == valeurCherchee Alors
      Retourner i ou true     // L'indice où la valeur a été trouvée
   Fin Si
Fin Pour
    Retourner -1 ou false     // Valeur non trouvée
Fin
```

Rappel : en java l'instruction return quitte complètement une fonction en retournant une valeur. L'instruction **break** ne quitte pas la fonction, mais ignore les lignes de codes suivantes pour quitter la boucle. Il existe également l'instruction **continue** qui, si elle est exécutée, ignore les lignes suivantes pour recommencer la boucle.

Exemple avec continue : dans ce programme toutes les valeurs du tableau sont mises à 0 sauf les valeurs 10 :

```
int A[]={10,15,56,15,20,10,65,0,15,10};
for (int i = 0; i < A.length; i++) {
    if(A[i]==10)
       continue;
    À[i]=0;     //la ligne n'est pas exécuté si A[i] vaut 10.
}
```

À faire : tester la fonction de recherche suivante.

```
/**
 *
 * @param A : le tableau A
 * @param v : la valeur cherchée
 * @return : boolean : true ==> v est trouvée dans A, false sinon
 */
public static boolean chercheValeur(int A[], int v) {
    for (int i = 0; i < A.length; i++) {
        if(A[i]==v){
            return true;
        }
    }
    return false;
}

public static void testChercheValeur() {
    int A[] = {10,20,30,40,50,60,70};
    boolean res = chercheValeur(A, 3);  //cherche dans A la valeur 3
    System.out.println("" + res);
}
```

EXERCICES

Écrire les fonctions suivantes, ainsi qu'un programme de test :

- Écrire une fonction permettant de vérifier si une valeur (v) est présente ou non dans un tableau. Si c'est le cas, cette fonction retournera la position de cette valeur (v), -1 sinon.
- Écrire une fonction permettant de vérifier si un tableau contient au moins une valeur comprise entre 15 et 20.
- Écrire une fonction permettant de vérifier si le tableau contient au moins une valeur ayant pour valeur le double de la valeur en case 0.
- Vérifier s'il existe une valeur (v) présente dans 2 cases côte à côte.