PASSAGE DE PARAMÈTRES

Exercice 1 Soit le code Java suivant :

```
class A {
    public static void main(String[] args){
        Scanner s=new Scanner(System.in);
        int a=s.nextInt();
        int b=s.nextInt();
        System.out.println(a+b);
    }
}

class B {
    static int somme(int a,int b){
        return a+b;
    }
}

class C {
    static int somme(){
        Scanner s=new Scanner(System.in);
        int a=s.nextInt();
        int b=s.nextInt();
        return a+b;
    }
}
```

Indiquer quelles sont les instructions qui correspondent à des appels de fonction, avec quels paramètres, quelle valeur de retour, quelles lignes correspondent à la définition d'une fonction, quels paramètres sont déclarés, quel type de retour... Y a-t-il une différence entre les deux fonctions somme ?

Exercice 2 Expliquer ce que produit l'exécution du programme suivant :

```
class Z {
    static void ajouterInt(int x){
        x=x+1;
    }
    static void ajouterTab(int[] tab){
        tab[0]=tab[0]+1;
    }
    public static void main(String[] ppp){
        int x=10;
        System.out.println(x);
        ajouterInt(x);
        System.out.println(x);
        int[] t=new int[3];
        t[0]=10;
        System.out.println(t[0]);
        ajouterTab(t);
        System.out.println(t[0]);
    }
}
```

Est-ce différent si on écrit :

```
static int ajouterInt(int x){
    x=x+1;
    return x;
}
```

Doit-on changer autre chose, pour que l'effet voulu soit réalisé ? Faire un schéma du contenu de la mémoire lorsque le point d'exécution est dans la fonction `ajouterInt`, dans la fonction `ajouterTab`.

Exercice 3 Écrire une méthode statique qui reçoit un tableau d'entiers en argument, et renvoie un (autre) tableau où tous les éléments ont été incrémentés d'une unité, sans modifier le tableau initial. Écrire une ligne de programme faisant appel à cette méthode.

Exercice 4 On définit le type suivant :

```
class IntRef{
    public int valeur;
    public IntRef(int n) {valeur = n;}
}
```

La fonction de Syracuse est définie pour un entier $n > 0$ par $f(n) = \begin{cases} n/2 & \text{si } n \text{ est pair,} \\ 3n+1 & \text{si } n \text{ est impair.} \end{cases}$

Écrire, en Java, une fonction statique `syracuse` qui reçoit en argument un `IntRef` correspondant à la valeur n et change sa valeur en $f(n)$. Écrire un programme qui demande à l'utilisateur une valeur initiale n et calcule les valeurs successives $f(n), f(f(n)), \ldots, f^p(n), \ldots$ jusqu'à obtenir 1*.
Le code suivant semble ne pas fonctionner. Pourquoi ? Comment le corriger ?

```
IntRef a = new IntRef(10);
IntRef b = new IntRef(3);
syracuse(b);
if (b==a) System.out.println("égaux");
else System.out.println("pas égaux");
```

Exercice 5 Dans certains langages de programmation comme le C, il n'y a pas véritablement de tableau à plusieurs dimensions. On simule alors leur fonctionnement en utilisant des tableaux à une dimension et une fonction permettant de calculer un indice.

- Donner un système simple permettant de représenter un tableau d'entiers de 10 lignes et 5 colonnes à l'aide d'un tableau à une seule dimension. Préciser comment accéder à l'élement de ligne `i` et de colonne `j`. Écrire les instructions correspondantes en Java.
- Écrire une fonction `elementAt` et recevant en entrée un tel tableau et deux entiers `i` et `j`. Cette fonction permettra de renvoyer l'élément de coordonnées `i`, `j` du tableau vu comme tableau de 10 lignes et 5 colonnes. Penser à vérifier que le tableau a la taille suffisante...
- Faire la même chose pour un tableau de dimension 3.
- On suppose maintenant qu'un tableau à deux dimensions est construit avec un tel codage, mais que sa taille dépend de valeurs `h` et `k` entrées par l'utilisateur (que l'on ne connaît pas quand on écrit le programme). Modifier la fonction `elementAt` de sorte qu'elle fonctionne toujours correctement.

Exercice 6 Écrire une fonction de nom barycentre, prenant en entrée un tableau de `Point` (*cf.* TD1) et renvoyant leur isobarycentre.

*Attention, rien ne garantit que le programme terminera car il n'y a pas de preuve qu'il existe bien un p vérifiant $f^p(n) = 1$ pour tout n. On l'a simplement vérifié expérimentalement jusqu'à 2^{62}.