PILES & APPLICATIONS

Exercice 1 (Partiel 2009) On se donne trois piles `p1`, `p2` et `p3`. La pile `p1` contient une suite de nombres entiers positifs. On suppose qu'il existe une classe `Pile` contenant les méthodes `estVide`, `sommet`, `empiler` et `depiler`.

1. Écrire un algorithme pour déplacer les entiers de `p1` dans `p2`, en inversant leur ordre.
2. Écrire un algorithme pour déplacer les entiers de `p1` dans `p2`, en conservant leur ordre.
3. Écrire un algorithme pour déplacer les entiers de `p1` dans `p2` de façon à avoir dans `p2` tous les nombres pairs au-dessous des nombres impairs.
4. Écrire un algorithme pour copier dans `p2` les nombres pairs contenus dans `p1`. Le contenu de `p1` après exécution de l'algorithme doit être identique à celui avant exécution. Les nombres pairs doivent être dans `p2` dans l'ordre où ils apparaissent dans `p1`.

Exercice 2 Certaines machines (virtuelles) devant effectuer des tâches limitées et relativement simples ont des capacités de gestion de la mémoire bien inférieures de celles des ordinateurs classiques. Parmi ces machines, on cite les *Automates à Pile* (AAP) dont la tâche est de vérifier qu'un mot donné (sur un alphabet quelconque) possède certaines propriétés.
La seule mémoire auxilliaire d'un AAP est une pile (dans laquelle on peut en principe stocker des objets de n'importe quel type). On n'a donc pas la possibilité de manipuler des variables entières pour implémenter des compteurs, par exemple. Les seules opérations de manipulation de la mémoire que l'on peut faire sont donc les opérations fondamentales de pile : empilement, dépilement, test de pile vide.

1. Écrire les différentes fonctions de manipulation d'une pile de caractères (en utilisant par exemple un tableau de caractères et un « indice » de sommet) : `empiler`, `depiler`, `estVide`, `sommet` et `afficher`.
2. Écrire un programme qui simule le comportement d'un AAP qui vérifie si une chaîne de caractères représente une expression bien parenthésée. Autrement dit, en ignorant tous les caractères qui ne sont pas des « `(` » ou des « `)` » (qui n'ont aucune importance pour le problème donné) la machine doit :
   – reconnaître comme « bons » les mots `()`, `()(())` ou `(((())())())`
   – reconnaître comme « mauvais » les mots `)(`, `()((` ou `(((()))))`.
3. Modifier le programme pour qu'il puisse reconnaître les expressions bien parenthésées contenant deux types de parenthèses (par exemple `( )` et `[ ]`).
4. *Écrire un programme qui simule le comportement d'un AAP qui vérifie si une chaîne de caractères (ne contenant que des lettres a et b) possède autant d'occurrences de a que d'occurrences de b. Par exemple $aababbbaab$.*
5. *Écrire un programme qui simule le comportement d'un AAP qui vérifie si une chaîne de caractères (ne contenant que des lettres a et b) possède un nombre d'occurrences de a égal au double du nombre d'occurrences de b. Par exemple $aabaab$ ou $babaaa$ (il y a 4 occurrences de a et 2 de b).*

Exercice 3 (Partiel 2011) On considère le problème des tours de Hanoi : déplacer n disques de diamètres différents d'une tour de départ 1 à une tour d'arrivée 2 en passant par une tour intermédiaire 3 et ceci en un minimum de coups, tout en respectant les règles suivantes :
- on ne peut déplacer plus d'un disque à la fois ;
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide (on suppose que cette règle est également respectée dans la configuration initiale).

On propose la solution suivante :

```
static void hanoi(int n){
    PileTab p=new PileTab();
    int[] a={1,2,3,0};  a[3]=n;
    p.empiler(a);
    while(!p.estVide()){
        int[] e=p.depiler();
        if (e.length == 2 || e.length == 4 && e[3] == 1)
            System.out.println(e[0] + "-->" + e[1]);
        else{
            a=new int[4]; a[0]=e[2]; a[1]=e[1]; a[2]=e[0]; a[3]=e[3] - 1;
            p.empiler(a);
            a=new int[2]; a[0]=e[0]; a[1]=e[1];
            p.empiler(a);
            a=new int[4]; a[0]=e[0]; a[1]=e[2]; a[2]=e[1]; a[3]=e[3]-1;
            p.empiler(a);
        }
    }
}
```

1. Écrire une classe `PileTab` qui permet de manipuler des piles de (références de) tableaux d'entiers comme dans la méthode `hanoi`.
2. Écrire une méthode `chaine` qui exprime le contenu d'une pile de type `PileTab` sous forme d'une chaîne de caractères (sans modifier la pile en question).
3. Expliquer, le plus précisément possible, ce que produit l'appel `hanoi(3)`.

Exercice 4 Il a été montré en cours qu'une expression arithmétique écrite en forme post-fixe peut facilement être évaluée à l'aide d'une pile. Cependant, la représentation « classique » des expressions arithmétiques est en forme infixe (les opérateurs sont plaçés *entre* et non *après* les deux opérandes, ce qui nécessite l'utilisation de parenthèses). On se propose donc de programmer la conversion d'une forme vers l'autre.
On commencera par convertir en forme infixe une chaîne de caractères `Exp` représentant une expression arithmétique écrite en forme post-fixe. Plus précisément, le programme devra lire une seule fois, de la gauche vers la droite, la chaine `Exp` et, en utilisant une pile de manière appropriée, affichera l'expression infixe correspondante. Pour simplifier l'analyse de la chaîne `Exp`, on pourra se limiter au cas où les opérandes sont toutes constituées d'une seule lettre et où les seuls opérateurs sont `+` et `*`.

On va écrire maintenant un programme qui effectue la conversion inverse : en lisant une seule fois, de la gauche vers la droite la chaîne de caractères `Exp` représentant une expression arithmétique écrite en forme infixe, afficher la même expression arithmétique écrite en forme post-fixe.

Voici quelques suggestions :
- Les opérandes sont directement affichées sans passer par la pile.
- La pile ne contient donc que des opérateurs.
- A chaque opérateur, sauf « `)` », est attribuée une priorité comme suit :
  - Opérateur `*` : priorité 2
  - Opérateur `+` : priorité 1
  - Opérateur `(` : priorité 0
- En fonction des priorités de l'opérateur couramment lu et de celui se trouvant au sommet de la pile, on devra effectuer des opérations différentes sur la pile (ne pas oublier la gestion des parenthèses fermantes), c'est à vous de déterminer lesquelles.

Vous pourrez vérifier votre l'algorithme en le simulant sur des exemples bien choisis.

```
A + B;
A + B * C;
( A + B ) * C;
A + B * C + D * E;
A + B * ( C + D ) * E;
```

Écrire enfin le programme correspondant.