TP 6 – Piles et fonctions récursives

Éléments de correction (provisoire)

Les codes source sont dans le fichier source associé

Présentation du problème

Une pile en informatique est un format de donnée similaire à une liste, mais, comme une pile d'assiettes, ne peut être manipulée que selon les cinq fonctions dont un listing est donnée ci-dessous :

```
# Fonctions propres aux piles construites Ã  partir des fonctions sur les listes

def creer_pile():
    return []

# Teste si une pile est vide (1 boulÃ©en oui, 0 boulÃ©en non)
def empty(P):
    return P==[]

# ajoute l'Ã©lÃ©ment x Ã  la pile P
def push(P,x):
    P.append(x) # on rappelle que liste.append(x) ajoute x Ã  la fin de liste

# supprime le dernier Ã©lÃ©ment de la pile
def pop(P):
    if empty(P): # utilise la fonction empty qui vient d'Ãªtre crÃ©Ã©e
        print("erreur : pile vide")
        return ("erreur")
    else:
        del P[-1] # supprime le dernier Ã©lÃ©ment de P (selon l'idÃ©e d'une liste cylique)
        return(P)

# indique quel est l'Ã©lÃ©ment au sommet de la pile P
def sommet(P):
    if empty(P):
        print("erreur : pile vide")
        return ("erreur")
    else:
        return P[-1]
```

1 Quelques exercices classiques sur les piles

Exercice 1 — Un peu de verlan

1. Écrire un programme prenant en entrée une chaîne de caractères et affichant ces caractères dans l'ordre inverse. On utilisera une pile.
2. Écrire un programme qui teste si une chaîne de caractères est un palindrome, c'est à dire un mot ou une expression qui est identique qu'on la lise de gauche à droite ou de droite à gauche.
   On voudrait utiliser le code écrit à la question **1.**, mais le programme ne retourne pas de données. L'idée est de faire une version du programme précédent qui au lieu de retourner un affichage retourne la pile des caractères inversés.

Exercice 2 — Hauteur de pile

1. Écrire une fonction `Hauteur(P)` qui calcule le nombre d'éléments dans une pile `P`.
   Lorsque cette fonction se termine la pile `P` devra avoir retrouvé son état initial.
2. Écrire une fonction `Copie(P1)` qui prend en entrée une pile `P1` et retourne une copie `P2` de la pile `P1` . Lorsque cette fonction se termine, la pile `P1` doit avoir retrouvé son état initial.

   IL faut considérer que la fonction ne garde pas la mémoire de `P1` au cours de son exécution, il faut donc remplir une autre pile pour la retrouver.

Exercice 3 — Bons parenthésages

1. Lire les expressions littérales suivantes, et à l'aide d'une pile, tester manuellement si elles sont ou non bien parenthésées (*on donnera les états successifs de la pile à chaque fois qu'un parenthèsage est repéré*).
   a. $[a + \{b/(c - d) + e/(f + g)\} - h]$

      États successifs de la pile à chaque fois qu'un parenthèsage est repéré :
      $\emptyset$ [[{ [{([{ [{([{ [$\emptyset$
      L'expression est bien parenthésée.
   b. $[a * \{b + c * (d - e)/f + \{g - h\}]$

      États successifs de la pile à chaque fois qu'un parenthèsage est repéré :
      $\emptyset$ [[{ [{([{ [{{ [{
      Problème : il est demandé de fermer un crochet alors qu'on ne peut fermer qu'une accolade.
      L'expression est mal parenthésée.
   c. $[a * \{b + c * (d + e) - f\} + g] - h]$

      États successifs de la pile à chaque fois qu'un parenthèsage est repéré :
      $\emptyset$ [[{ [{([{ [$\emptyset$
      Problème : il est demandé de fermer un crochet alors que tous les parenthésages ont déjà été fermés.
      L'expression est mal parenthésée.

2. Écrire une fonction qui prend en argument une chaîne de caractères représentant une expression mathématique parenthésée et qui teste si cette expression mathématique est bien parenthésée.

On peut faire un copier/coller des expressions de la question **1.** pour tester le programme.

2 Écriture en polonaise inverse

La notation polonaise inverse est une écriture des opérations utilisée par certaines calculatrices (souvent anciennes). On l'appelle aussi notation postfixée car elle consiste à écrire les opérateurs après leurs opérandes. Bien que peu usuelle pour les occidentaux, cette notation permet d'écrire des calculs algébriques sans avoir besoin de recourir à des parenthésage. De plus, avec de l'habitude la notation polonaise inverse permet de calculer plus vite que la notation usuelle.

Exemples

- l'expression $3 + 7$ s'écrit $3\ 7\ +$
- L'expression $(3 + 7) \times 2$ s'écrit $3\ 7\ +\ 2\ \times$ mais aussi $2\ 3\ 7\ +\ \times$

Pour calculer en notation polonaise inverse on utilise une pile des opérandes et des opérateurs. Quand on évalue une expression postfixée, on empile les nombres dès qu'on les voit apparaître. Par contre quand on lit un opérateur :

- on désempile les deux nombres au sommet de la pile des opérandes ;
- on effectue l'opération sur ces deux nombres ;
- on empile le résultat du calcul.

Exercice 4 — Implémenter une fonction `Evaluation (L)` qui prend en entrée une liste contenant soit des nombres soit des opérateurs (décrivant une expression algébrique en notation polonaise inverse), et qui retourne la valeur de l'évaluation de cette expression.
Pour simplifier, on supposera que seuls deux opérateurs apparaissent : $+$ et $*$.

Pour écrire une expression usuelle (dite ***Infix***) en notation postfixée, on utilise une pile des parenthèses et opérateurs :

- On commence par mettre une parenthèse `(` dans la pile
- lorsqu'on lit un nombre, on le met tout de suite dans l'expression postfixée
- Lorsqu'on lit un opérateur `*` ou une parenthèse `(` on le met dans la pile
- Lorsqu'on lit un opérateur $+$ et que le sommet de la pile n'est pas `*` on empile l'opérateur `+`
- Losqu'on lit un opérateur $+$ et que le sommet de la pile est `*` on désempile la pile d'opérateurs et on met * dans l'expression postfixée
  On continue cette action de désempilage jusqu'à ce que le sommet de la pile ne soit plus `*`, quand c'est le cas, on empile `+`
- Lorsqu'on lit une parenthèse `)` on désempile la pile (et on met les caractères désempilés dans l'expression posfixée) jusqu'à ce que le l'on ait désempilé une parenthèse `(`.
  Attention : on **ne met pas** de parenthèse dans l'expression postfixée !

Exemple
On convertit l'expression

$$(a + (b + c * d + b * a) * b * c)$$

Les opérations effectuées sont en dernière page.

Exercice 5 — Écrire une fonction `Postfix(L)` qui transforme une liste contenant une expression écrite en notation Infix et retourne une liste contenant la conversion de cette expression en notation postfix.

3 Tours de Hanoï

Le jeu des tours de Hanoï est constitué de trois tiges (représentées par des piles). Sur l'une des tiges sont empilés n disques de tailles différentes, de sorte que chaque disque soit posé sur un disque plus grand que lui.
Le but du jeu est de déplacer tous les disques de la tige de départ vers une autre tige, en déplaçant un seul disque à la fois, et en ne posant jamais un disque sur un plus petit que lui.
écrire une fonction `Hanoi(T1, T2, T3,n)` qui indique la suite des mouvements à effectuer pour déplacer n disques, initialement posés sur la tige $T1$ portant le numéro 1, vers la tige $T3$ portant le numéro 3 (en utilisant la tige $T2$ portant le numéro 2).

Par exemple, on devra avoir

```
>>> Hanoi("A","B","C",3)
Deplacer un disque depuis  A  vers  C
Deplacer un disque depuis  A  vers  B
Deplacer un disque depuis  C  vers  B
Deplacer un disque depuis  A  vers  C
Deplacer un disque depuis  B  vers  A
Deplacer un disque depuis  B  vers  C
Deplacer un disque depuis  A  vers  C
```

Remarque : La résolution de cette énigme peut se programmer sans utiliser de fonction récursive, mais alors la solution est compliquée à concevoir et à réaliser.
L'algorithme ci-dessous réalise la tâche avec une grande économie de lignes et de réflexion (la réflexion doit cependant être de qualité pour comprendre le fonctionnement de cet algorithme récursif !).

L'idée est que si on peut réaliser la tâche avec $n-1$ disques, on peut la réaliser avec n disques. En effet, à partir de la position de départ :

1. *déplaçons les $n-1$ disques supérieurs sur la deuxième tige*
2. *déplaçons le plus grand disque de la première à la troisième tige*
3. *déplaçons les $n-1$ autres disques sur la troisième tige*

*Si $n=1$, on fait directement l'étape **3**.*
Si $n=2$, on fait les étapes 1.2.3. de façon simple.
Si $n=3$

- *l'étape **1** consiste à mettre en oeuvre l'algorithme pour $n=2$, mais en permutant les rôles des tiges 2 et 3*
- *l'étape **2** est toujours la même*
- *l'étape **3** consiste à mettre en oeuvre l'algorithme pour $n=2$, mais en permutant les rôles des tiges 1 et 2*

Si $n=4$

- *l'étape **1** consiste à mettre en oeuvre l'algorithme pour $n=3$, mais en permutant les rôles des tiges 2 et 3*

- *l'étape **2** est toujours la même*
- *l'étape **3** consiste à mettre en oeuvre l'algorithme pour $n = 3$, mais en permutant les rôles des tiges 1 et 2*

...

```
def Hanoi(T1,T2,T3,n):
    if n > 0:
        Hanoi(T1,T3,T2,n-1) # Etape 1
        print("Deplacer un disque depuis ", T1, " vers ", T3) # Etape 2
        Hanoi(T2,T1,T3,n-1)  # Etape 3
```

Si on entre `Hanoi('A','B','C',1)`, voici ce qui se passe :

- *exécute `Hanoi('A','B','C',1)`*
  - *exécute `Hanoi('A','C','B',0)` qui ne fait rien car $n = 0$*
  - *affiche : `Déplacer un disque depuis A vers C`*
  - *exécute `Hanoi('B','A','C',0)` qui ne fait rien car $n = 0$*
- *Fin du programme*

Si on entre `Hanoi('A','B','C',2)`, voici ce qui se passe :

- *exécute `Hanoi('A','B','C',2)`*
  - *exécute `Hanoi('A','C','B',1)`*
    - *exécute `Hanoi('A','B','C',0)` qui ne fait rien car $n = 0$*
    - *affiche : `Déplacer un disque depuis A vers B`*
    - *exécute `Hanoi('C','A','B',0)` qui ne fait rien car $n = 0$*
  - *affiche : `Déplacer un disque depuis A vers C`*
  - *exécute `Hanoi('B','A','C',1)`*
    - *exécute `Hanoi('B','C','A',0)` qui ne fait rien car $n = 0$*
    - *affiche : `Déplacer un disque depuis B vers C`*
    - *exécute `Hanoi('A','B','C',0)` qui ne fait rien car $n = 0$*
- *Fin du programme*

Si on entre `Hanoi('A','B','C',3)`, voici ce qui se passe :

- *exécute `Hanoi('A','B','C',3)`*
  - *exécute `Hanoi('A','C','B',2)`*
    - *exécute `Hanoi('A','B','C',1)`*
      - *exécute `Hanoi('A','C','B',0)` qui ne fait rien car $n = 0$*

- *affiche :* `Déplacer un disque depuis A vers C`
- *exécute* `Hanoi('B','A','C',0)` *qui ne fait rien car* $n = 0$

⊙ *affiche :* `Déplacer un disque depuis A vers B`

⊙ *exécute* `Hanoi('C','A','B',1)`

- *exécute* `Hanoi('C','B','A',0)` *qui ne fait rien car* $n = 0$
- *affiche :* `Déplacer un disque depuis C vers B`
- *exécute* `Hanoi('A','C','B',0)` *qui ne fait rien car* $n = 0$

◦ *affiche :* `Déplacer un disque depuis A vers C`

◦ *exécute* `Hanoi('B','A','C',2)`

⊙ *exécute* `Hanoi('B','C','A',1)`

- *exécute* `Hanoi('B','A','C',0)` *qui ne fait rien car* $n = 0$
- *affiche :* `Déplacer un disque depuis B vers A`
- *exécute* `Hanoi('C','B','A',0)` *qui ne fait rien car* $n = 0$

⊙ *affiche :* `Déplacer un disque depuis B vers C`

⊙ *exécute* `Hanoi('A','B','C',1)`

- *exécute* `Hanoi('A','C','B',0)` *qui ne fait rien car* $n = 0$
- *affiche :* `Déplacer un disque depuis A vers C`
- *exécute* `Hanoi('B','A','C',0)` *qui ne fait rien car* $n = 0$

▶ *Fin du programme*

$$\sim$$