1. Généralités sur les piles

Définition Wikipedia

Une pile (en anglais stack) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (LIFO pour Last In, First Out), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés. Le fonctionnement est celui d'une pile d'assiettes : on ajoute des assiettes sur la pile, et on les récupère dans l'ordre inverse, en commençant par la dernière ajoutée.

Définition pratique

Concrètement, une pile est une structure de données munie des fonctions suivantes :

- pile_vide() : renvoie une pile vide
- empiler(*p, x*) (ou push) : ajoute un élément *x* sur la pile *p*
- dépiler(*p*) (ou pop) : enlève un élément de la pile *p* et le renvoie.
- est_vide(*p*) : renvoie True si la pile est vide, False sinon.

Ces quatre fonctions sont les seules indispensables pour définir la structure de pile, mais on peut disposer d'autres fonctions, par exemple :

- nombre_elements(*p*) : renvoie le nombre d'éléments de la pile.
- lire_sommet(*p*) (ou peek) : renvoie l'élément de tête sans le dépiler.
- vider_pile(*p*) : dépile tous les éléments.
- afficher_pile(*p*) : affiche le contenu de la pile *p*

Exercice : programmer ces fonctions à partir des quatre primitives.

Remarque

- Les opérations depiler, empiler et lire_sommet s'effectuent en principe en temps constant (complexité en O(1)). nombre_elements et vider_pile peuvent être en temps constant ou linéaire selon leur implémentation.
- Le plus souvent, nos piles ne seront pas limitées en taille (autrement que par la mémoire disponible), mais elles peuvent l'être, si elles sont construites à partir de tableau de taille constante par exemple.

Exercice *Dans les questions suivantes, les seules fonctions autorisées sur les piles sont celles définies précédemment.*

a) Ecrire les fonctions dup, swap, rot, agissant sur une pile *p* (passée en paramètre) définies comme suit :
   - dup duplique l'élément de tête de la pile
   - swap permute les deux éléments de tête de la pile
   - rot effectue une permutation circulaire des trois éléments de tête
   - rotn effectue une permutation circulaire des *n* éléments de tête

b) Ecrire deux fonctions, copier_pile et inverser_pile, de paramètre une pile *p*, qui renvoient respectivement une copie de *p* et une autre pile contenant les éléments de *p* dans l'ordre inverse sans modifier *p*.

2. Implémentation en Python

Nous utiliserons des listes pour travailler sur les piles et les fonctions suivantes.

```
def pile_vide():
    return []

def empiler(p, x):
    p.append(x)

def est_vide(p):
    return p==[]

def depiler(p):
    return p.pop()
```

Il est bien sûr intéressant de programmer nombre_elements avec la fonction len() plutôt que de compter le nombre d'éléments en dépilant.

3. Applications

Dans un navigateur web, une pile sert à mémoriser les pages Web visitées. L'adresse de chaque nouvelle page visitée est empilée et l'utilisateur dépile en cliquant le bouton « Précédent ».

La fonction « Annuler » (Undo) d'un traitement de texte mémorise les modifications apportées au texte dans une pile.

Les appels récursifs d'une fonction provoquent l'écriture dans une pile de l'adresse mémoire de l'instruction qui a appelé la fonction, et des valeurs des

paramètres et des variables définies par la fonction. Dans un langage non récursif, on peut simuler la récursivité à l'aide d'une pile.

4. Compléments : les files

Définition Wikipedia

Une file (queue en anglais) est une structure de données basée sur le principe du Premier entré, premier sorti (FIFO : First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés. Le fonctionnement ressemble à une file d'attente : les premières personnes à arriver sont les premières personnes à sortir de la file.

Primitives

- file_vide() : renvoie une file vide
- enfiler(*f*, *x*) (ou enqueue) : ajoute un élément *x* dans la file *f*.
- défiler(*f*) (ou dequeue) : renvoie le prochain élément de la file, et le retire de la file.
- est_vide(*f*) : renvoie True si la file est vide, False sinon.

Applications

- mémoriser temporairement des transactions qui doivent attendre pour être traitées
- mémoire tampons (buffers)
- serveurs d'impression (qui traitent les requêtes dans l'ordre dans lequel elles arrivent)

5. Compléments : implémentation fonctionnelle (récursive)

Dans notre implémentation des piles, les fonctions empiler et depiler modifient la pile sur laquelle on les applique. On dit que nos piles sont mutables. En programmation fonctionnelle, on ne modifie jamais l'état d'un objet, mais on crée de nouveaux objets. Ces objets sont immuables. Les fonctions empiler et depiler ne modifient alors plus la pile existante mais renvoient une autre pile.

Implémentation fonctionnelle

```
def empiler(pile, e):
    return (e, pile)

def depiler(pile):
    return pile[1], pile[0]
```

pile = (3, (2, (1, ())))
pile, elt = depiler(pile) affecte (2, (1, ())) à pile et 3 à elt.
pile = empiler(pile, 4) affecte (4, (3, (2, (1, ())))) à pile.

Recherche du minimum d'une pile mutable

```
def minimum(pile):
    pile_temp = pile_vide()
    mini = depiler(pile)
    empiler(pile_temp, mini)
    while not est_vide(pile):
        val = depiler(pile)
        empiler(pile_temp, val)
        if val < mini:
            mini = val
    while not est_vide(pile_temp):
        empiler(pile, depiler(pile_temp))
    return mini
```

Recherche du minimum d'une pile immuable

```
def minimum(pile):
    pile, mini = depiler(pile)
    while not est_vide(pile):
        pile, val = depiler(pile)
        if val < mini:
            mini = val
    return mini
```

Exercices

Exercice 1

Dans cet exercice, les seules fonctions que l'on utilisera avec des piles sont pile_vide(), empiler(p, x), depiler(p), est_vide(p).

Un programmeur a écrit un petit script gérant l'inscription à une activité. Les personnes intéressées entrent leurs coordonnées sur un ordinateur. Leurs noms, adresses et téléphones sont stockés au fur et à mesure de leur inscription dans des piles nommées pile_nom, pile_adresse, pile_phone.

Ecrire une fonction compile(pile_nom, pile_adresse, pile_phone) qui renvoie une pile contenant les données des trois piles précédentes sous forme de tuples du type (*nom, adresse, téléphone*). La fonction affichera un message d'erreur et renverra une pile vide si les trois piles n'ont pas le même nombre d'éléments.

Exercice 2 : Implémentation d'une file à l'aide de deux piles

On dispose de deux piles A et B, muni des fonctions habituelles, et on souhaite les utiliser comme une file. L'ajout des éléments se fait toujours dans la pile A. On retire les éléments de la pile B. Si celle-ci est vide, on transfère d'abord tous les éléments de la pile A vers la pile B.

a. Programmer les fonctions enfiler et defiler suivant le principe précédent.
b. On utilise cette file pour enfiler puis défiler *n* éléments. Quel est la complexité de l'algorithme correspondant ?

Exercice 3

Écrire une fonction d'entête **def** parenthesage(chaine): qui, à partir d'une chaine de caractères contenant des parenthèses, crochets et accolades, vérifie si le parenthésage est bon.

Exercice 4

Écrire une fonction switch renvoyant une copie d'une pile passée en argument en inversant ses éléments du sommet et du bas, sans modifier la pile de départ.

Exercice 5 : Tri d'une pile

1. Recherche sur papier

On dispose d'une pile p = [2, 4, 1, 5, 3] et de deux piles q et r. Transférer, en les triant, tous les éléments de la pile p dans la pile q, en vous aidant si besoin de la pile r.

3		
5		
1		
4		
2		
p	q	r

2. Implémentation

Ecrire une fonction tri_pile de paramètre une pile *p* contenant des entiers renvoyant la pile *p* triée. Cette fonction n'utilisera pour seules variables que deux autres piles *q* et *r*, munies de leurs fonctions habituelles.

Vous ne pourrez pas programmer une solution que vous ne connaissez pas… ! Il est souvent utile de chercher une méthode sur un exemple avant d'essayer de la programmer.