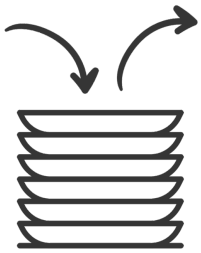


©AdaCore

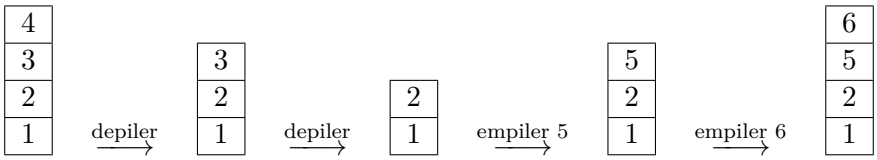
Les piles

1 Généralités

Les *piles* sont une structure de données dans laquelle on stocke les éléments dans leur ordre d'arrivée et on y accède dans l'ordre inverse. C'est donc un fonctionnement analogue à celui d'une pile d'assiettes : la dernière posée sur la pile sera la première qui en sortira. En anglais on parle de traitement LIFO (Last In, First Out).
Les deux opérations de base sur les piles sont :

- une fonction `push` ou `empiler` qui ajoute un élément au-dessus de la pile ;
- une fonction `pop` ou `depiler` qui retire l'élément au sommet de la pile.

Exemple.



Les piles sont couramment utilisées en informatique.

- L'historique des adresses visitées dans un navigateur internet : le bouton *précédent* dépile la dernière adresse visitée.
- Dans un éditeur de texte, chaque action est empilée dans une pile et le bouton *annulation* dépile la dernière modification ;
- Lors de l'exécution d'un programme, les adresses mémoires des fonctions appelées sont mises dans une pile (qui a une taille finie).

2 Implémentation en Python

En Python, les piles seront implémentées par des listes, le sommet de la pile correspondant au dernier élément de la liste. Ainsi `[1, 2, 3]` correspond à la pile $\begin{array}{|c|}\hline 3 \\ \hline 2 \\ \hline 1 \\ \hline \end{array}$.

Attention : bien qu'on utilise des listes pour représenter les piles, on n'utilisera que les fonctions propres aux piles (les deux opérations de base citées précédemment). Par exemple, pour la pile `p = [1, 2, 3]`, l'opération `p[0]` n'est pas licite car elle revient à « regarder » l'élément du bas de la pile qui n'est normalement pas accessible.
Ainsi, pour faire clairement la différence, dans la suite du cours, on notera `stack` pour désigner les piles dans les signatures des fonctions. Par ailleurs, on notera `object` les éléments que l'on peut mettre dans la pile : `int`, `float`, `str`, `list`, `tuple`, etc.

La première opération concernant les piles est la création d'une nouvelle pile (vide).

```python
def creer_pile() -> stack:
    """ Création d'une pile vide. """
    return []
```

La seconde est l'opération d'empilage d'un élément donné `v` sur une pile `p`. Cette fonction ne renvoie rien puisqu'il n'y a pas de `return`, elle agit par *effet de bord* sur la pile.

```python
def empiler(p: stack, v: object) -> None:
    """ Empile v sur la pile p. """
    p.append(v)
```

Enfin, l'opération de dépilage. Plus précisément, celle-ci enlève l'élément du dessus de la pile `p` et renvoie sa valeur.

```python
def depiler(p: stack) -> object:
    """ Dépile le sommet de p et le renvoie. """
    return p.pop()
```