| Cours | Cours 5.1 | TSI1 |
|---|---|---|
| | Piles | Période 5 |
| | | 30min |

Dans un programme où les données traitées arrivent au fur et à mesure, il peut être commode d'envisager des listes à accès particulier : celles où l'ajout / le retrait d'un élément se fait uniquement aux extrémités.

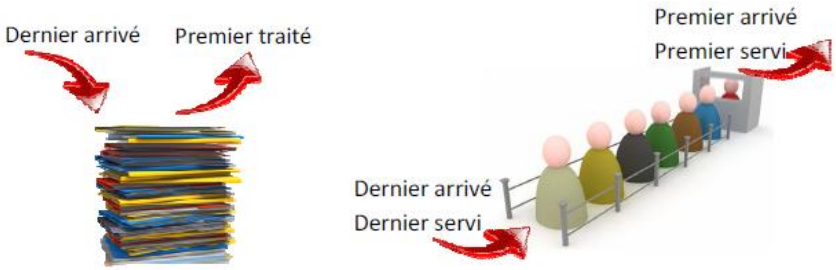



**Figure 1 : Pile de travail / File d'attente**

# 1 Pile et file

"LIFO" (Last In First Out) : on ajoute/retire à une seule extrémité.
La liste modélise alors une pile : on ajoute au sommet **push** et on retire au sommet **pull**.

Exemples :

- mémorisation des pages web visitées dans un navigateur (précédent = pull, suivant = push),
- fonction définie par récurrence : pile des appels récurrents.

"FIFO" (First In First Out) : on ajoute à une extrémité de la liste et on retire à l'autre. La liste modélise alors une file.

Exemple : gestion des tâches d'une imprimante.

Dans ce cours d'informatique, on se limitera à l'utilisation des piles.

# 2 Pile

## 2.1 Fonctions élémentaires d'un pile

Avant de voir comment implémenter concrètement une pile, il peut être intéressant de décrire l'ensemble des opérations que cette structure de données doit permettre :

- **creer_pile()** : fonction qui renvoie une pile initialement vide.
- **depiler(p)** : dépile et renvoie le sommet de la pile p (si non vide).
- **empiler(p,v)** : empile la valeur v sur la pile p (si non pleine).

On veut également disposer de fonctions qui donnent des caractéristiques de la pile sans la modifier :

- **taille(p)** : renvoie le nb d'éléments dans la pile p.
- **est_vide(p)** : indique si la pile p est vide.
- **sommet(p)** : renvoie la valeur au sommet de la pile p.

Exemple d'utilisation pour une pile de taille 4 :

```
p=creer_pile 4)
empiler(p,'A')
empiler(p,'B')
empiler(p,'C')
depiler(p)
depiler(p)
empiler(p,'D')
```

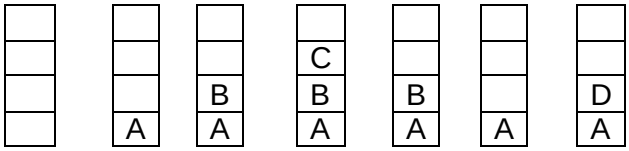



**Figure 2 : Evolution de la pile en fonction des instructions (présentation verticale : pile)**

## 2.2 Comparaison entre une pile et une liste

**Liste** : une liste a des opérations d'accès à n'importe quel élément qui la compose (à partir de l'indice de l'élément par exemple).

**Pile** : une pile ne permet que l'accès au dernier élément stocké et l'ajout en dernière place.
C'est une structure utilisée pour la gestion des priorités de certains programmes (les derniers programmes lancés sont prioritaires : ce type de structure permet notamment le bon fonctionnement des fonctions récursives).
Une structure de pile interdit par construction les opérations sur les éléments intermédiaires.

# 3 Implémentation des piles

## 3.1 Pile à capacité non bornée

Sous Python, les piles seront simulées à partir des listes. Il faudra veiller à respecter le protocole d'empilement et dépilement.
Une pile à capacité non bornée à la taille du nombre d'éléments stockés dans la pile.

Les méthodes associées aux listes permettent d'obtenir des piles à capacité non bornée :

- empiler grâce à **L.append(v)** permet d'ajouter **v** à droite de la liste **L**,
- dépiler grâce à **L.pop()** récupère le dernier élément de la liste **L** et le retire de **L**.

Remarque : v = v + [1] est équivalent à v.append(1) mais la dernière instruction est plus rapide car elle ne traite que l'ajout d'une valeur alors que la concaténation réaffecte l'ensemble de la liste.
La concaténation est ainsi plus complexe en temps et en mémoire.

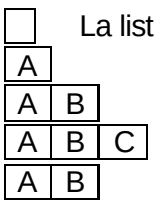
Les avantages de la pile non bornée sur la pile bornée sont :

- de ne pas être limitée dans le nombre de valeurs à stocker,
- de ne pas avoir besoin d'enregistrer le nombre d'éléments stockés dans la pile (la taille de la pile est la taille de la liste),

L'inconvénient est que la pile non bornée peut prendre un volume important (pas d'information sur le fait que les appels des fonctions récursives ne se terminent pas).

Exemple de fonctionnement :

```
p1 = creer_pile()
empiler(p1,'A')
empiler(p1,'B')
empiler(p1,'C')
depiler(p)
```




La liste p1 est vide : pas d'élément stocké.

**Fonctions agissant sur la pile :**

```
def creer_pile():
    return [ ]
```

```
def depiler(p):
    if len(p)>0 :
        return p.pop()
```

```
def empiler(p,v):
    p.append(v)
    return p
```

**Fonction d'information sur la pile :**

```
def taille(p):
    return len(p)
```

```
def est_vide(p):
    return len(p)==0
```

```
def sommet(p):
    if len(p)>0 :
        return p[-1]
```

**Effet de bord**

La liste locale **p** d'une fonction **f(p)** est un alias de la liste passée en entrée lors de l'appel de la fonction par l'instruction **f(p1)**, par exemple. La liste originale **p** est donc modifiée par les fonctions : il n'est donc pas indispensable de renvoyer, en sortie de la fonction, la liste modifiée. C'est ce que l'on appelle l'**effet de bord**

Exemple :

```
def empiler(p,v):
    p.append(v)
```

```
p1=[0,1,2]
empiler(p1,3)
>>> p1          # la modification de p a impacté directement la variable globale p1
[0,1,2,3]       # car p est l'alias de p1 dans la fonction empiler.
```

Cet effet de bord prévu en python sur les listes permet d'améliorer l'efficacité du traitement des listes de grande taille. Cette implémentation n'est pas saine car elle perturbe la lisibilité de la fonction : pas de sortie apparente car c'est l'entrée qui est modifiée.

Pour modifier une liste passée en entrée d'une fonction, sans qu'elle modifie directement la liste globale, il faut passer par une variable locale.

Exemple :

```
def ajouter(L,v):
    L1=L[0:]    # L1 : locale, est affectée par les indices pour ne pas créer un alias
    L1.append(v)
    return L1
```

```
p1=[0,1,2]
L2 = ajouter(p1,3)
>>> p1
[0,1,2]
>>> L2
[0,1,2,3]
```

Pour la pile, il est plus simple de conserver la liste **p** comme un alias de **p1** car les modifications faites dans les fonctions **depiler()** et **empiler()** doivent être répercutées sur la liste passée en entrée.

## 3.2 Pile à capacité bornée

Pour créer une pile de capacité bornée, il suffit de définir un seuil et changer l'expression de la fonction empiler.

```
seuil = 1000

def empiler(p,v):
    if len(p)>=seuil :
        print ("Overflow : dépassement de capacité")
    else :
        p.append(v)
    return p
```

Ressources :
Damien Broizat
Patrick Beynet –UPSTI-