

**Objectif :** Choisir une structure de données adaptée à la situation à modéliser.

## 1 Problématique

Le processeur peut adopter plusieurs stratégies pour exécuter l'enchaînement des processus. Selon l'algorithme utilisé (*First Come First Served*, *Shortest Job First...*) la structure adoptée pour stocker la liste des tâches a une importance fondamentale.

Quelles structures de données adopter pour implémenter les algorithmes d'ordonnancement ?

## 2 Des structures héritées de la liste chaînée

### 2.1 Pile

#### 2.1.1 Présentation

Les piles (*stack* en anglais) sont fondées sur le principe du *dernier arrivé premier sorti* (*Last In First Out*). L'image classique est la pile d'assiette.

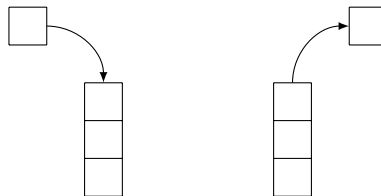


FIGURE 1 – Empiler - dépiler

#### 2.1.2 Interface

Les éléments sont *empilés* ou *dépilés* en respectant la règle du *LIFO*. La pile doit alors proposer ces fonctionnalités. Une interface classique de pile d'éléments de type noté  $T$  est la suivante :

- **créer\_pile()**  $\rightarrow$  **Pile()** : crée une pile vide
- **est\_vide()**  $\rightarrow$  **bool** : renvoie *True* si la pile est vide, *False* sinon.
- **empiler(e : T)**  $\rightarrow$  **None** : ajoute un élément  $e$  au sommet de la pile.
- **depiler()**  $\rightarrow$  **T** : retire et renvoie l'élément du sommet de la pile.

#### 2.1.3 Implémentation

Les principes utilisés pour créer une liste chaînée sont repris ici.

##### Activité 1 :

1. Créer la classe **Nœud** identique à la classe *Maillon* déjà utilisée pour créer une liste chaînée.
2. Créer une classe **Pile**. Elle possédera un attribut *contenu* initialisé à *None*.
3. Créer les méthodes proposées dans l'interface.

### 2.2 File

#### 2.3 Présentation

Les files (*queue* en anglais) sont fondées sur le principe du *premier arrivé premier sorti* (*First In First Out*). L'image classique est la file d'attente.

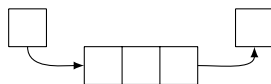


FIGURE 2 – Enfiler - défiler

### 2.3.1 Interface

Les éléments sont *enfilés* ou *défilés* en respectant la règle du *FIFO*. La file doit alors proposer ces fonctionnalités. Une interface classique de file d'éléments de type noté  $T$  est la suivante :

- **creer\_file()**  $\rightarrow$  **File()** : crée une file vide
- **est\_vide()**  $\rightarrow$  **bool** : renvoie *True* si la file est vide, *False* sinon.
- **enfiler(e : T)**  $\rightarrow$  **None** : ajoute un élément  $e$  à l'arrière de la file.
- **defiler()**  $\rightarrow$  **T** : retire et renvoie l'élément de l'avant de la file.

### 2.3.2 Implémentation

Ici encore une classe *Nœud* sera utilisée pour créer les éléments

#### Activité 2 :

1. Récupérer la classe **Nœud**.
2. Créer une classe **File**. Elle possédera deux attributs : *tete* et *queue*.
3. Créer les méthodes proposées dans l'interface.

## 3 Ordonnancement

Reprenons certains algorithmes d'ordonnancement.

#### Activité 3 :

1. Rappeler le principe du *First Come First Served*. Quelle structure semble adaptée à cet algorithme ?
2. Même question pour le *Round Robin*.