# P31 Développement efficace - épisode 4

## 1 - Des piles

Une pile (`stack`) est une structure de données qui permet de stocker des éléments en contraignant les opérations d'ajout de de retrait à respecter la règle du **dernier entré, premier sorti** (LIFO : *Last In, First Out*).

Les seules opérations possibles sur une pile sont :

- `push` : ajouter un élément au sommet de la pile
- `top` : accéder à l'élément au sommet de la pile
- `pop` : supprimer l'élément au sommet de la pile

Les opérations n'ayant lieu que d'un côté de la pile (son sommet), il est aisé et efficace de les implémenter avec un `vector`.

> ***Créer les fichiers `containers/stack.h` et `containers/stack.c`.***
> ***Définir la structure `stack` et l'ensemble des fonctions suivantes :***

- initialisation et libération :

```
void stack_init(stack *s, size_t value_size);
void stack_free(stack *s);
```

- accès aux propriétés courantes :

```
size_t stack_size(stack const *s);
size_t stack_value_size(stack const *s);
```

- manipulation des données :

```
// copie la mémoire pointée par value dans un nouvel élément au sommet de la pile
void stack_push(stack *s, void const *value);
// copie la valeur contenue dans l'élément du sommet de la pile
// à l'adresse pointée par value
void stack_top(stack const *s, void *value);
// copie la valeur contenue dans l'élément du sommet de la pile
// à l'adresse pointée par value, puis supprime cet élément
void stack_pop(stack *s, void *value);
```

## 2 - Un algo sur les piles

Le problème des tours de Hanoï est un problème classique de récursivité qui se résout de manière élégante avec des piles.

Le problème consiste à déplacer un ensemble de disques de tailles différentes d'une tour de *départ* à une tour d'*arrivée* en se servant d'une tour *intermédiaire*, tout en respectant les règles suivantes :

- on ne peut déplacer qu'un seul disque à la fois
- on ne peut poser un disque que sur un disque plus grand (ou sur une tour vide)

Les disques peuvent être représentés par des entiers, la taille du disque étant égale à la valeur de l'entier. Les tours peuvent être représentées par des piles.

> ***Créer les fichiers `algos/hanoi_towers.h` et `algos/hanoi_towers.c`.***
> ***Définir la fonction suivante :***
>
> ```
> // n est le nombre de disques à déplacer de la pile start à la pile end
> void hanoi_towers(int n, stack *start, stack *inter, stack *end);
> ```

***Indication***: la fonction s'écrit de manière récursive en moins de 10 lignes.

## 3 - Des files

Une file (`queue`) est une structure de données qui permet de stocker des éléments en contraignant les opérations d'ajout de de retrait à respecter la règle du **premier entré, premier sorti** (FIFO : *First In, First Out*).

Les seules opérations possibles sur une file sont :

- `enqueue` : ajouter un élément en queue de la file
- `front` : accéder à l'élément en tête de la file
- `dequeue` : supprimer l'élément en tête de la file

Les opérations ayant lieu des deux côtés de la file (sa tête et sa queue), il est aisé et efficace de les implémenter avec une liste chaînée.

> ***Créer les fichiers `containers/queue.h` et `containers/queue.c`.***
> ***Définir la structure `queue` et l'ensemble des fonctions suivantes :***
> *Indication* : il sera utile de maintenir un pointeur sur le noeud de queue de la liste chaînée.

- initialisation et libération :

```
void queue_init(queue *q, size_t value_size);
void queue_free(queue *q);
```

- accès aux propriétés courantes :

```
size_t queue_size(queue const *q);
size_t queue_value_size(queue const *q);
```

- manipulation des données :

```
// copie la mémoire pointée par value dans un nouvel élément à la queue de la file
void queue_enqueue(queue *q, void const *value);
// copie la valeur contenue dans l'élément en tête de la file à l'adresse pointée par value
void queue_front(queue const *q, void *value);
// copie la valeur contenue dans l'élément en tête de la file à l'adresse pointée par value, puis su
void queue_dequeue(queue *q, void *value);
```

## 4 - Des files avec des piles ?

Est-il possible d'implémenter une file avec deux piles ? Si oui, quel serait l'espace mémoire nécessaire pour une telle implémentation avec nos piles à base de `vector` ?

> ***Réfléchir à une potentielle manière de faire, puis, si la réflexion est concluante, l'écrire.***