

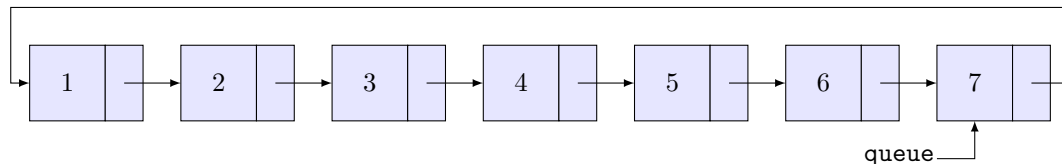
# TP N°5 - IMPLÉMENTATIONS D'UNE FILE

Récupérer le fichier `tp05_debut.ml` présent sur le site de classe et qui contient la définition des types pour les deux premiers exercices et une proposition de trame pour le dernier exercice.

## 1 Implémentation d'une file

## 2 Implémentation à l'aide d'une liste chaînée circulaire

Une liste chaînée circulaire est une liste chaînée dont le dernier élément pointe vers le premier.



Afin de représenter une telle liste, on définit les types suivants :

```
type 'a cellule = {valeur: 'a ; mutable suivant: 'a cellule};;
type 'a liste = Nil | Cellule of 'a cellule;;
```

Nous allons utiliser une liste chaînée circulaire pour représenter une file, ce qui amène à définir le type suivant :

```
type 'a file = {mutable queue : 'a liste};;
```

On pointera sur le dernier élément de la file, autrement dit le dernier élément à être entré. Le premier élément à sortir sera donc l'élément suivant.

Par exemple, la liste représentée ci-dessous représentera la file 1 – 2 – 3 – 4 – 5 – 6 – 7.

**Q1** Écrire une fonction `creer_file` : `unit -> 'a file` qui renvoie une file vide.

**Q2** Écrire une fonction `enfiler` : `'a -> 'a file -> unit` qui ajoute un élément dans la file.

**Q3** Écrire une fonction `defiler` : `'a file -> 'a` qui supprime et renvoie l'élément en début de file.

**Q4** Tester les fonctions précédentes :

- créer une pile vide;
- lui ajouter les entiers de 1 à 5;
- supprimer et afficher 3 éléments de la file;
- lui ajouter les entiers 6 et 7;
- vider la file en affichant ses éléments.

### 2.1 Implémentation avec deux piles

On considère désormais le type `file` suivant :

```
type 'a file = {
  entree : 'a Stack.t;
  sortie : 'a Stack.t
};;
```

L'ajout d'un élément se fera dans la pile `entree`, la suppression d'un élément dans la pile `sortie`.

On rappelle quelques fonctions du module `Stack` :

- `Stack.create` : `unit -> 'a Stack.t` crée une pile vide;

- `Stack.push : 'a -> 'a Stack.t -> unit` ajoute un élément dans une pile;
- `Stack.pop : 'a Stack.t -> 'a` supprime l'élément en haut d'une pile et le renvoie;
- `Stack.is_empty : 'a Stack.t -> bool` teste si une pile est vide.

- Q5** Écrire une fonction `creer_file : unit -> 'a file` qui renvoie une file vide.
- Q6** Écrire une fonction `enfiler : 'a -> 'a file -> unit` qui ajoute un élément dans la file.
- Q7** Écrire une fonction `defiler : 'a file -> 'a` qui supprime et renvoie l'élément en début de file.
- Q8** Tester vos fonctions avec le même protocole que dans l'exercice précédent.

### 3 Pour les plus rapides : Feu de forêt

On considère un damier représentant une forêt partiellement en feu. Au départ, une seule case est en feu, puis le feu se propage. La probabilité  $p$  que le feu se propage d'une case en feu aux cases voisines est constante. On considère que le damier est un carré de côté 400 et que l'incendie naît au centre du carré. L'algorithme de propagation utilise une structure de file :

Allumer le centre du damier.  
Insérer le centre dans la file des cases en attente (précédemment vide).  
Tant que la file n'est pas vide,  
    extraire une case de la file.  
    Pour chacune des cases voisines,  
        Si la case n'est pas en feu,  
            tirer au sort la propagation avec la probabilité  $p$ ,  
            Si le feu se propage,  
                allumer la case  
                l'ajouter à la file.

- Q9** Compléter la fonction `propagation` qui simule graphiquement l'évolution de l'incendie. Elle prend en argument la probabilité  $p$ ; chaque case sera représentée par un pixel.
- Q10** Tester différentes valeurs de  $p$  et trouver expérimentalement la plus petite valeur de  $p$  pour laquelle les chances de propagation du feu jusqu'aux bords semblent non nulles.