TP n° 10 : Piles et files

Les piles (*stack* en anglais) et les files (*queue* en anglais) sont deux exemples de structures de données fondamentales en informatique. Il faut bien les connaître. Vous reverrez les piles en cours de tronc commun d'informatique l'année prochaine.

Ces deux structures de données diffèrent par l'ordre d'accès aux éléments :

- Dans une *pile*, la valeur extraite est la dernière valeur à avoir été placée dans la pile. On parle de structure LIFO (*Last In First Out*).
- Dans une *file*, l'élément que l'on retire de la file est le premier à y être entré. On parle de structure FIFO (*First In First Out*).

Question 1

Faire un schéma et donner deux ou trois exemples[a] d'utilisation de ces structures dans la vie courante.

a. Originaux et drôles si possible — et dans ce cas appelez-nous pour les partager.

Avant d'implémenter de plusieurs manières ces structures, nous allons les utiliser. Remarquez qu'il n'y a même pas besoin d'avoir implémenté les structures pour pouvoir définir les fonctions qui les utilisent, à condition d'avoir bien spécifié leur signature (type et opérations valables). Dans ce TP, nous allons utiliser les modules `Stack` et `Queue` que vous avez définis à l'aide du cours.

EXERCICE 1 *Taille d'une pile et d'une file*

Écrire les fonctions qui permettent de connaître la taille (le nombre d'éléments) d'une pile et d'une file. Quelle est la complexité de votre approche ? Que faudrait-il faire pour pouvoir écrire des fonctions efficaces ?

EXERCICE 2 *Copies*

On dispose d'une pile de copies de deuxième année, triées par noms d'élèves, qui proviennent de MP ou de MP*. Comment procéder pour transformer cette pile en une pile dans laquelle toutes les copies de MP sont situées au-dessus de celles de MP*, en conservant l'ordre relatif des copies, c'est-à-dire que les copies de MP et de MP* sont toujours bien triées (sans les trier à nouveau, évidemment) ? On ne pourra utiliser que des piles (de copies).

On propose les types :

OCAML
```
type classe = MP | MPstar
type copie = {nom : string; note : int; classe : classe}
```

Écrire une fonction `separe : copie Stack.t -> unit` qui réordonne une pile de copies suivant ce principe.

EXERCICE 3 *Rotations*

Écrire une fonction de type `'a Stack.t -> ()` qui réalise la rotation d'une pile, c'est-à-dire que l'élément au sommet se retrouve tout en bas de la pile.

$$\begin{vmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{vmatrix} \longrightarrow \begin{vmatrix} a_1 \\ \vdots \\ a_n \\ a_0 \end{vmatrix}$$

On pourra utiliser une pile auxiliaire. Quelle est sa complexité ?

Écrire la fonction qui réalise la rotation inverse. Faire de même avec une file au lieu d'une pile.

EXERCICE 4 *Mots bien parenthésés*

On considère des mots sur l'alphabet $\{(, [,),]\}$. Par exemple $(||((]($ et $()[()]$ sont deux mots. On représente les mots par une liste de caractères (`type mot == char list`). Le mot vide ne comportant aucune lettre est noté ε. Il est donc représenté par la liste vide. On dit qu'un mot est *bien parenthésé* s'il peut être réduit au mot vide en supprimant successivement des paires adjacentes de parenthèses de même type. Par exemple le mot $([()[]])()$ est bien parenthésé car :

$$([()[]])() \to ([[]])() \to ([])() \to ()() \to () \to \varepsilon$$

Écrire une fonction `est_bien_parenthese : mot -> bool` qui vérifie si un mot est bien parenthésé.