Exercice 1 : Pile ou file

1. On considère une **pile** sur laquelle on effectue 12 opérations : 6 fois empiler, et 6 fois dépiler. L'ordre de ces 12 opérations n'est pas a priori précisé, tant qu'il est légal.

   Les éléments sont empilés dans cet ordre : A, B, C, D, E, F.

   Parmi les ordres suivants, lesquels sont possible pour les valeurs renvoyées par le fonction dépiler ?

   Quand c'est possible, indiquer quel ordre des opérations peut y conduire, et quand ce n'est pas possible, expliquer pourquoi.

   (a) ABCDEF
   (b) FEDCBA
   (c) CDBEAF
   (d) CDEABF

2. Même question mais on a cette fois-ci une **file** sur laquelle on effectue 12 opérations : 6 opérations enfiler et 6 opérations défiler.

   Les éléments sont enfilés dans cet ordre : A, B, C, D, E, F.

Exercice 2 : Utilisation des piles pour parser une expression

On reprend une question similaire à l'exercice 2 du TP4 : on se donne t un tableau de chaînes de caractères [1] de la forme "+", "-", "*", "/", ou d'un entier écrit en décimal, et on veut construire l'arbre représentant une expression arithmétique dont t est l'écriture **suffixe**.

Pour celà, le fichier `exo2.ml` contient déjà les définitions des types et des fonctions de conversion utiles.

1. Que proposez-vous de mettre dans la pile pour résoudre ce problème de façon itérative ?
2. Écrire une fonction `suffixe_vers_arbre : string array -> arbre_arith` qui réalise cette conversion. La tester.

1. On parle de *lexème* pour des chaînes de caractères qui correspondent à une unité lexicale d'une formule.

Exercice 3 : Implémentation des files mutables de deux façons différentes

1. (a) Implémenter, selon les algorithmes vus en cours, des files **mutables** comme un enregistrement nommé contenant deux listes mutables, comme ci-dessous :

```
type 'a file_1 = {mutable entree : 'a list; mutable sortie : 'a list}
```

   (b) *(Chez vous)* Faire de même avec des files **immuables** :

```
type 'a file_2 = {entree : 'a list; sortie : 'a list}
```

2. Une autre implémentation possible des files utilise les tableaux de façon circulaire : on crée un enregistrement nommé où, en plus du tableau, on maintient la position de l'entrée et de la sortie de la file. Si l'entrée de la file dépasse l'indice maximal du tableau, alors on la fait retourner à l'indice 0.

   Par exemple, si on utilise un tableau de taille 5, et qu'on fait les opérations suivantes :

   - Enfiler A
   - Enfiler B
   - Enfiler C
   - Défiler (on obtient A)
   - Défiler (on obtient B)
   - Enfiler D

   Alors la file sera représentée par la structure suivante (où les cases vides représentent des valeurs non utilisées) :

		C	D	

   Entrée : indice 4 ; Sortie : indice 2

   Si on effectue ensuite les opérations suivantes :

   - Enfiler E
   - Enfiler F
   - Enfiler G

   Alors la file sera représentée par :

F	G	C	D	E

   Entrée : indice 2 ; Sortie : indice 2

   Il sera alors impossible d'enfiler un nouvel élément dans la file, la capacité maximale étant atteinte.

   (a) Comment distinguer une file vide d'une file pleine ?

   (b) Écrire le type `'a file_3` en utilisant des tableaux circulaires, en tenant compte de la question précédente.

   (c) On veut implémenter les fonctions qui définissent les files. Attention, pour créer un tableau on doit connaître le type de ses valeurs et avoir une valeur par défaut. La fonction `creer_3` prendra également un argument supplémentaire indiquant la capacité de la file.

   On aura donc `creer_3 : int -> 'a -> 'a file`, avec la capacité en 1er argument et la valeur par défaut en deuxième. Écrire cette fonction.

   (d) Écrire les fonctions `est_vide_3` et `est_pleine_3`.

   (e) Écrire les fonctions `enfiler_3` et `defiler_3`. La fonction `enfiler_3` renverra une erreur si la file est pleine.

3. Écrire une fonction `test_1 : int -> unit` qui étant donné un entier n, effectue n opérations de file (idéalement, choisies au hasard) sur une file mutable du type défini à la question 1a.

   Faire de même avec une fonction `test_3 : int -> unit` qui fait la même chose sur une file mutable du type défini à la question 2b, qu'on prendra de capacité au moins n.

   Comparer l'efficacité des deux implémentations.