TD 5

Piles et files

Pour ce TD nous utiliserons pour les piles le module `Stack` de Ocaml, et le module `Queue` pour les files.
On dispose des primitives suivantes :

Module Stack	
val create : unit -> 'a t	Return a new stack, initially empty.
val push : 'a -> 'a t -> unit	push x s adds the element x at the top of stack s.
val pop : 'a t -> 'a	pop s removes and returns the topmost element in stack s, or raises Stack.Empty if the stack is empty.
val is_empty : 'a t -> bool	Return true if the given stack is empty, false otherwise.

Module Queue	
val create : unit -> 'a t	Return a new queue, initially empty.
val add : 'a -> 'a t -> unit	add x q adds the element x at the end of the queue q.
val pop : 'a t -> 'a	take q removes and returns the first element in queue q, or raises Queue.Empty if the queue is empty.
val is_empty : 'a t -> bool	Return true if the given queue is empty, false otherwise.

(Source : https ://ocaml.org/api/Stack.html ,https ://ocaml.org/api/Queue.html)

Exercice 1 :

1. Écrire la fonction `tete` qui prend une pile p et renvoie l'élément en tête de la pile, sans éliminer cette élément de la pile.

   ```
   tete : 'a Stack.t -> 'a
   ```

2. Écrire la fonction `creation_pile` qui prend en argument une liste et retourne la pile dont les éléments sont stockés dans l'ordre de leur lecture dans la pile. Exemple :

   ```
   creation_pile : 'a list -> 'a Stack.t
   ```

3. Écrire la fonction `taille` qui prend en argument une pile, et renvoie le nombre d'élément qu'elle contient. (la pile ne doit pas finir vide à l'issue d'appel de la fonction.)

   ```
   taille : 'a Stack.t -> int
   ```

Exercice 2 :

1. Écrire la fonction `creation_file` qui prend en argument une liste et retourne la file dont les éléments sont stockés dans l'ordre de leur lecture dans la liste.

   ```
   creation_file : 'a list -> 'a Queue.t
   ```

2. Écrire la fonction `taille` qui prend en argument une file, et renvoie le nombre d'élément qu'elle contient. (la file ne doit pas finir vide à l'issue d'appel de la fonction.)

3. Écrire la fonction `separation` qui prend en argument une file d'entier et qui sépare dans deux files les nombres paires et impaires.

Exercice 3 : **Bon parenthésage :**
On cherche à vérifier la bonne imbrication des parenthèses, crochets ou accolades (Klammer en allemand) dans une phrase.
On pose :

```
let ouvrante = ['[' ; '(' ; '{' ];;
let fermante = [ ']' ; ')' ; '}'];;
```

1. Écrire la fonction `associe` qui prend deux caractères et renvoie le fait que les Klammer sont associés.

2. Écrire une fonction `bon_parenthesage` qui prend en argument une chaine de caractère représentant une expression algébrique, et vérifie à l'aide d'une pile le bon parenthésage de l'expression.
   Exemple :

   ```
   let phrase_exemple ="(1+2[x+2] -3 {5x+3})";;
   bon_parenthesage phrase_exemple;;
   - : bool = true
   let phrase_exemple_2 ="(1+2[x+{ 2] -5x+3})";;
   bon_parenthesage phrase_exemple_2;;
   - : bool = false
   ```

Exercice 4 : On veut utiliser piles pour évaluer des expressions arithmétiques écrites en syntaxe post fixée.
On définit tout d'abord l'ensemble $\mathcal{T}$ des termes constituant ces expressions. La définition est donnée par induction, dans l'ensemble de tous les mots écrits à partir de l'alphabet E (noté E^*) constitué des entiers relatifs et des caractères $+, -, \times, /$ par :

- $\forall x \in \mathbb{Z}$, $x \in \mathcal{T}$,
- $\forall t_1 \in \mathcal{T}$, $\forall t_2 \in \mathcal{T}$, $t_1 t_2 + \in \mathcal{T}$,
- $\forall t_1 \in \mathcal{T}$, $\forall t_2 \in \mathcal{T}$, $t_1 t_2 - \in \mathcal{T}$,
- $\forall t_1 \in \mathcal{T}$, $\forall t_2 \in \mathcal{T}$, $t_1 t_2 \times \in \mathcal{T}$,
- $\forall t_1 \in \mathcal{T}$, $\forall t_2 \in \mathcal{T}$, $t_1 t_2 / \in \mathcal{T}$.

Pour une meilleur lisibilité des expressions post-fixées, les lettres seront séparées d'un point. Ainsi nous pouvons donner comme exemple :
$t_{ex} = 6.2. + .3.1. \times .-$, on a bien $t_{ex} \in \mathcal{T}$.
On appelle "symbole" les caractères $+, -, \times, /$.

Question 1 :
Pour un élément $t \in \mathcal{T}$, on note $s(t)$ le nombre de symbole dans l'écriture de t, et $n(t)$ le nombre d'entier dans l'écriture de t.
Montrer que :

$$\forall t \in \mathcal{T} \quad s(t) = n(t) - 1$$

Question 2 :
Pour $t \in \mathcal{T}$, on appelle préfixe de t toutes séquences u de E^* pour lesquelles il existe une séquences v de E^* avec $t = uv$.
Dans l'exemple précédent, $u = 6.2. + .3$ est un préfixe de t_{ex}.
Montrer que, si $t \in \mathcal{T}$ et si u est un préfixe non vide de t, alors :

$$s(u) < n(u)$$

Question 3 :
Les termes suivants appartiennent-ils à $\mathcal{T}$?

- $t_1 = 2.3.4. + .\times$
- $t_2 = 6.5.7. + . - .\times$
- $t_3 = 12.45.53.10. - . \times .2.3. + ./.\times$
- $t_4 = 1.56.10.5. + . - .4./. + .8. \times . - .7.2.+$

Question 4 :
Pour représenter les expressions arithmétiques, nous créons les types `symbole` Symbole et `lettre`, qui permettrons de traiter les entiers et les symboles comme étant de même types :

```
type symbole = Plus | Moins | Fois | Div;;

type lettre = Val of int | Sym of symbole;;
```

Un mot sera une suite de lettres, et donc sera représenté par une liste de lettres :

```
type mot = lettre list;;
```

Les expressions arithmétiques post-fixées seront des mots (bien entendu, tous les mots ne sont pas des expressions arithmétiques post-fixées).
Pour évaluer un mot, considéré a priori comme une expression arithmétique post-fixée, on se sert d'une pile initialement est vide. On prend alors successivement les lettres du mot.

- Si c'est un entier (de la forme `Val(a)`), on empile la valeur a sur la pile ;
- Si c'est un symbole :
  - on prend les deux entiers du haut de la pile.
  - Si la pile ne contient pas deux entiers, on rend une erreur : le mot initial n'était pas une expression arithmétique.

  - A partir de l'opération représentée par le symbole, et des deux valeurs dépilées, on calcule le résultat de l'opération, résultat que l'on empile.

- Lorsque le mot a été entièrement lu, il ne doit rester qu'un seul entier dans la pile : le résultat de l'évaluation de l'expression. On dépile ce résultat, la pile doit être vide.
  Si elle ne l'est pas, on rend une erreur, sinon, on rend le résultat.
  Attention : pour les opérations non commutatives (soustraction, division), le premier argument se trouve *sous* le second dans la pile.

1. Sur feuille, appliquer à la main l'algorithme décrit ci-dessus pour l'expression post-fixée suivante :
   $t_{ex} = 6.2. + .3.1. \times .-$

2. Écrire une fonction `eval_expr_arith` qui prend en argument un mot et l'évalue en utilisant l'algorithme donné ci-dessus.

   ```
   eval_expr_aritht : mot -> int;;
   ```