Piles fonctionnelles

Exercice 1. Considérons le type `pile` et ses primitives telles qu'indiquées dans un fichier `mli`

```
type 'a pile = Empty | P of 'a * 'a pile
val is_empty : 'a pile -> bool
val create : unit -> 'a pile
val append : 'a -> 'a pile -> 'a pile
val peek : 'a pile -> 'a
val dup : 'a pile -> 'a pile
val pop : 'a pile -> 'a pile
val swap : 'a pile -> 'a pile
val rotate : 'a pile -> 'a pile
```

— La fonction `is_empty` renvoie un bouléen indiquant si la pile est vide ;
— `create` crée une pile vide ;
— `append` ajoute un élément en tête de pile ;
— `peek` renvoie la valeur de l'élément au somet de la pile (une exception est soulevée si ce n'est pas possible) ;
— `dup` duplique l'élément au sommet de la pile :

```
# let p = P(1,P(2,P(3,Empty))) in
dup p;;
  - : int pile = P (1, P (1, P (2, P (3, Empty))))
```

une exception est soulevée si ce n'est pas possible ;
— `pop` retire le sommet de la pile :

```
# let p = P(1,P(2,P(3,Empty))) in
pop p;;
  - : int pile = P (2, P (3, Empty))
```

une exception est soulevée si ce n'est pas possible ;
— `swap` échange les deux premiers éléments de la pile

```
# let p = P(1,P(2,P(3,P(4,P(5,Empty))))) in
swap p;;
  - : int pile = P (2, P (1, P (3, P (4, P (5, Empty)))))
```

une exception est soulevée si ce n'est pas possible ;
— `rotate` échange le premier et le dernier élément de la pile :

```
# let p = P(1,P(2,P(3,P(4,P(5,Empty))))) in
rotate p;;
  - : int pile = P (5, P (2, P (3, P (4, P (1, Empty)))))
```

une exception est soulevée si ce n'est pas possible ;
Implanter toutes ces fonctions en utilisant `failwith` pour déclencher les exceptions.