université PARIS-SACLAY | FACULTÉ DES SCIENCES D'ORSAY

PFA

L3 Info, L3 Mag Info

TP 6

Présentation

Le but du TP est de se familiariser avec les traits impératifs d'OCaml. On continuera aussi d'utiliser les foncteurs. Le TP est composé de deux exercices chacun situé dans un sous-répertoire (`exo1` et `exo2`). Il est fortement conseillé de se placer dans le répertoire contenant ces deux sous-répertoire. Chacun des exercices à la même structure :

— un fichier `main.ml` et d'autres fichiers `.ml` contenant le code OCaml que vous devez compléter

— un fichier `dune` contenant les instructions de compilation

La commande `dune build` construit les deux exécutables `exo1/main.exe` et `exo2/main.exe`

1 Fonctions simples sur les tableaux

Écrire les fonctions suivantes en respectant les contraintes demandées. Chaque fonction devra être accompagnée de tests.

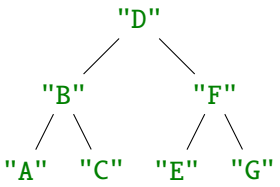
1. Une fonction `min_array : ('a -> 'a -> int) -> 'a array -> 'a`, qui renvoie le plus petit élément d'un tableau donné en second paramètre, selon la fonction de comparaison donnée en premier paramètre. Votre fonction **doit** utiliser une boucle **`for`** et lever l'exception prédéfinie `Not_found` si le tableau est vide.
2. Une fonction `is_sorted : ('a -> 'a -> int) -> 'a array -> bool` qui renvoie **`true`** si les éléments du tableau donné en second paramètre sont ordonnés selon la fonction de comparaison passée en premier paramètre. La fonction **doit** s'arrêter sur la première paire d'éléments consécutifs non ordonnés. Votre fonction ne doit pas utiliser d'exception ni de fonction récursive et doit renvoyer **`true`** pour le tableau vide.
3. Une fonction `reverse : 'a array -> unit` qui renverse un tableau en place (le tableau initial est modifié). Vous ne devez pas utiliser de fonction récursive
4. Une fonction `average : float array -> float option`. La fonction renvoie `None` si le tableau est vide et `Some m` sinon, où `m` la moyenne des valeurs du tableau. Votre fonction ne doit utiliser ni boucle ni récursion mais un itérateur du module `Array`.
5. Une fonction `shuffle : 'a array -> unit` qui mélange un tableau en place, en utilisant l'algorithme de Knuth[1]. On pourra d'abord écrire une fonction `rand_interval : int -> int -> int` telle que `rand_interval a b` renvoie un entier aléatoire dans l'intervalle `[a; b[` (`b` exclus). Cette fonction pourra utiliser la fonction OCaml `Random.int : int -> int` qui telle que `Random.int b` renvoie un entier entre 0 et `b` exclus.
6. Une fonction `rotate : 'a array -> unit` qui effectue une rotation d'une case vers la droite de tableau donné en argument. Ainsi si `t` est un tableau `[|1;2;3;4|]`, après appel à `rotate t`, le tableau vaut : `[|4;1;2;3|]`.

1. `https://en.wikipedia.org/wiki/Knuth_shuffle`

2 Arbres, Fifo et Lifo

2.1 Arbres binaires

Le fichier `tree.ml` contient une définition de type d'arbre binaire. Le fichier `main.ml` crée l'arbre de test



La définition de type est celle classique d'arbre binaire (sans stocker d'information supplémentaire). Compléter le code de la fonction `left_most : 'a tree -> 'a` qui renvoie la valeur la plus à gauche dans l'arbre et lève une exception si l'arbre est vide. La fonction ne doit pas être récursive mais utiliser une boucle **`while`**.

Indication : on pourra écrire des fonctions simples permettant de tester si l'arbre est vide, s'il a un fils gauche non vide ...

2.2 Parcours d'arbre

On donne ensuite, dans le fichier `tree.ml` un foncteur `Iter`. Ce dernier prend en argument un module dont le type est donné par la signature `S` se trouvant dans le fichier `defn.ml`. Cette signature est celle d'une collection *mutable* dans laquelle on peut ajouter et retirer des éléments un à un.

La fonction `iter` du foncteur `Iter` effectue ensuite un parcours itératif de l'arbre donné au moyen d'une boucle **`while`**.

Lire la signature `Defn.S` et le code de `Iter.iter`.

2.3 Fifo

Le module `fifo.ml` implémente une structure de pile, comme un enregistrement contenant deux champs mutables, l'un stockant une liste de valeurs et l'autre la taille.

1. implémenter les fonctions manquantes
2. dans `main.ml` instancier le foncteur `Iter` et vous servir de la fonction`iter` pour afficher les nœuds de l'arbres. Quel parcours est effectué ?

2.4 Lifo

Le module `lifo.ml` implémente une structure de file. Cette dernière est constituée d'un type auxiliaire `'a cell` représentant des listes chaînées mutables de valeurs. À la différence des listes d'OCaml, ces listes permettent de modifier en place la suite de la liste.

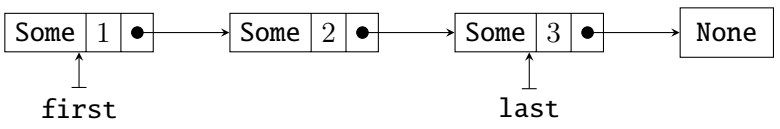
Le type des files est ensuite un enregistrement contenant :

— un champ `first` pointant vers `None` si la file est vide et `Some c` sinon, où `c` contient le prochain élément à sortir de la file.

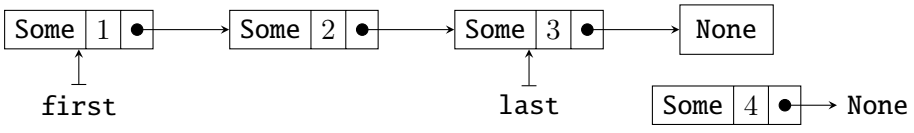
— un champ `last` pointant vers la dernière cellule insérée dans la file.

— un champ `size` contenant la longueur de la liste.

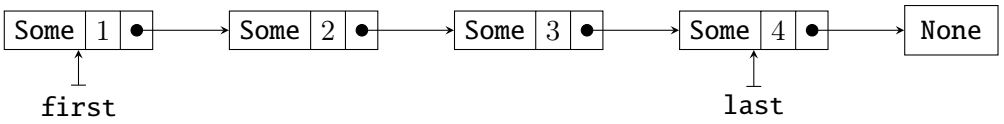
Le fonctionnement d'une file est le suivant. Supposons que la file contienne déjà les valeurs 1, 2, 3 ajoutées dans cet ordre.



Pour ajouter la valeur 4, il faut allouer une nouvelle cellule :

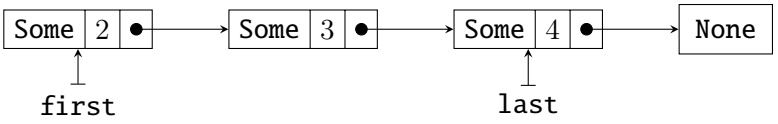


Il faut ensuite faire pointer le champ `next` de la cellule pointée par `last` vers cette nouvelle cellule, puis mettre `last` à jour pour pointer vers cette nouvelle cellule.



Si on souhaite maintenant retirer un élément de la liste, il suffit de renvoyer la valeur pointée par le champ `first`, puis de mettre à jour ce dernier pour pointer vers l'élément suivant :

1 (sorti de la file)



1. implémenter les fonctions manquantes de `lifo.ml`. On fera très attention, au cas de l'ajout dans une file vide ou à la suppression du dernier élément d'une file, qui doivent être gérés spécialement.
2. dans `main.ml` instancier le foncteur `Iter` et vous servir de la fonction`iter` pour afficher les nœuds de l'arbres. Quel parcours est effectué ?