TP n°14 – Implémenter les piles et les files

1 Implémentation des piles par le module Stack en Ocaml

Dans cette partie, on va utiliser le module `Stack` qui implémente pour nous une pile en Ocaml.

Les primitives ont le type suivant, où `'a Stack.t` est le type d'une pile :

- `Stack.create : unit -> 'a Stack.t` qui crée une pile vide
- `Stack.is_empty : 'a Stack.t -> bool` qui teste si une pile est vide
- `Stack.push : 'a -> 'a Stack.t -> unit` qui ajoute un élément
- `Stack.pop : 'a Stack.t -> 'a` qui retire et renvoie l'élément le plus neuf
- `Stack.top : 'a Stack.t -> 'a` qui renvoie sans retirer l'élément le plus neuf

- **Q1.** Créer une pile vide p.
- **Q2.** Ajouter les entiers 2,4,7,1, dans cet ordre.
- **Q3.** On donne la fonction suivante pour afficher le contenu d'une pile (sinon on ne peut pas le voir.

```
let affiche_pile p =
  let pp = Stack.create () in
  while not (Stack.is_empty p) do
    let x = Stack.pop p in
    Printf.printf "%d " x;
    Stack.push x pp
  done;
  Printf.printf "\n";
  while not (Stack.is_empty pp) do
    Stack.push (Stack.pop pp) p
  done;;
```

  Vérifier que votre pile contient bien ce qu'elle devrait.
- **Q4.** Dépiler une fois et vérifier le contenu de votre pile ainsi que l'élément dépilé.

2 Implémenter une file avec deux piles en Ocaml

Dans cette section on utilisera deux piles mutables du module `Stack` pour créer un file mutable, selon le principe expliqué en cours (une qui sert d'entrée et une qui sert de sortie).

- **Q5.** Définir un type `'a file` approprié.
- **Q6.** Écrire une fonction `creer_file : unit -> 'a file` qui renvoie une file vide.
- **Q7.** Écrire une fonction `est_vide_file : 'a file -> bool` qui vérifie si une file est vide.
- **Q8.** Écrire une fonction `enfile : 'a file -> 'a -> unit` qui enfile un élément dans une file.
- **Q9.** Écrire une fonction `defile : 'a file -> 'a` qui défile un élément de la file et renvoie la valeur défilée.
- **Q10.** Écrire une fonction `regarde_file : 'a file -> 'a` qui renvoie le premier élément de la file sans modifier la file. On fera attention à ce que la file ne soit pas vide.

3 Utiliser les piles et les files ?

On dit qu'un texte est bien parenthésé si à chaque parenthèse ouvrante "(" est associé une parenthèse fermante ")", la parenthèse fermante apparaissant *après* la parenthèse ouvrante.

Par exemple "Bonjour (à (tous))" est bien parenthésé mais "Bonjour (à (tous)" ne l'est pas, ni "Bonjour)à (tous)(".

- **Q11.** En utilisant l'algorithme proposé en cours, écrire une fonction `bien_parenthese: string -> bool` qui détermine si un texte est bien parenthésé.

En Ocaml, les chaines de caractères se comportent comme des tableaux de caractères, mais avec des notations différentes :

- Pour accéder au caractère i de la chaine `chaine` on écrit `chaine.[i]`
- Pour calculer la longueur, on écrit `String.length chaine`
- Pour concaténer deux chaînes de caractères on utilise l'opérateur `^` : `let c3 = c1 ^ c2;;` concatène les chaines `c1` et `c2`.

En mathématiques, les expressions arithmétiques comme $(2+7)*5$ sont écrites avec les opérandes et opérateurs placés de manière dite **infixe** : les opérateurs, tels $-, +, *$ et $/$ sont placés *entre* leurs opérandes.

La notation **postfixe** consiste en revanche à mettre les opérateurs *après* les opérandes. L'expression $(2 + 7) * 5$ s'écrit alors $2\ 7\ +\ 5\ *$. En effet les opérandes de $+$ sont 2 et 7 et celles de $*$ sont $2 + 7$ et 5. Il s'avère que cette écriture ne nécessite pas de parenthèses.

- **Q12.** Comment se réécrivent les expressions arithmétiques $(3 + 5 * 6)/7$ puis $(2 - 5 * 5 + 3) * (3 + 4)$?
- **Q13.** Que remarque t'on vis à vis de l'ordre de priorité des opérations ? Des parenthèses ?
- **Q14.** Deviner ce que peut être la notation **préfixe** des expressions arithmétiques.
- **Q15.** Réecrire les expressions arithmétiques précédentes mais en notation préfixe cette fois.
- **Q16.** En utilisant une pile, écrire une fonction Ocaml `evaluation_postfixe: string -> int` qui étant donné une expression arithmétique en notation postfixe, sous la forme d'une chaine de caractères (contenant uniquement des chiffres, les opérateurs $+ - */$ et des espaces), renvoie le résultat de l'expression. Par exemple, pour $2\ 7\ +\ 5\ *$ elle renverra 45.

  Dans cette question, vous pouvez considérer que les nombres apparaissant dans l'expression postfixe sont inférieurs à 9.

  Je recommande d'essayer de trouver par vous-même mais si vous êtes bloqués regardez à la fin du TP.
- **Q17.** (Bonus) Modifier la fonction précédente pour permettre d'évaluer des expressions qui contiennent des nombres supérieurs à 10.
- **Q18.** En utilisant une pile on peut également évaluer les expressions en notation préfixe. Comment faire ? L'implémenter en Ocaml.

Comme dernière application, on propose d'étudier l'inversion des éléments d'une pile.

- **Q19.** Écrire en Ocaml une fonction prenant en entrée une pile et utilisant une file pour renvoyer une pile avec les mêmes éléments qu'en entrée, mais dans l'ordre inverse.

4 Implémenter une file par un tableau circulaire en C

On définit le type `typedef struct File {int* contenu; int debut; int fin;} file;` pour réaliser une implémentation de file par un tableau circulaire en C.

Pour rappel un tableau circulaire est un tableau tel que la première et la dernière case sont considérées comme voisines.

- **Q20.** Réaliser l'implémentation des primitives de la file en C. Attention : on manipulera uniquement des `file*` pour garantir la mutabilité de la structure.

Une aide pour la question 16 :

On crée une pile p d'entiers dans laquelle on va stocker tous les résultats intermédiaires.

Pour chaque caractère dans l'expression :

- Si c'est un espace, passer au suivant
- Si c'est un chiffre, empiler le chiffre
- Si c'est une opération, dépiler les deux premiers nombres de la pile, effectuer l'opération dessus et empiler le résultat.

À la fin la pile ne devrait contenir qu'une seule valeur qui est le résultat.