IF111 - Algorithmes et structures de données-TD2 : Piles, Files et Listes

Jonathan Narboni, Rohan Fossé

jonathan.narboni@labri.fr, rfosse@labri.fr

Exercice 1

On dispose d'une pile contenant des nombres entiers. Écrire un algorithme qui permet d'afficher le contenu de la pile. On veut voir s'afficher dans l'ordre le sommet, le second élément, et ainsi de suite. On prendra soin que la pile, à l'issue de l'affichage, contienne les mêmes éléments *dans le même ordre*. L'algorithme utilise une pile auxiliaire. Comment modifier cet algorithme pour afficher les éléments dans l'ordre inverse ?

Exercice 2

On veut écrire des algorithmes pour manipuler des files et des piles. Les algorithme peuvent utiliser des piles et des files auxiliaires.

- On se donne une pile $P1$ contenant des entiers positifs. Ecrire un algorithme pour déplacer les entiers de $P1$ dans une pile $P2$ de façon à avoir dans $P2$ tous les nombres pairs en dessous des nombres impairs. L'algorithme prend en entrée $P1$ et $P2$ et utilise $O(1)$ mémoire auxiliaire.
- Ecrire un algorithme qui renverse une file. Faites de même pour une pile.
- En utilisant seulement des files, écrire un algorithme qui prend en entrée une file F contenant les éléments triés en ordre croissant (la tête est le plus petit élément de la file) et un élément e à enfiler. L'algorithme retournera la file F contenant l'élément e tout en préservant l'ordre croissant des éléments.

Exercice 3

On veut implémenter une file en utilisant deux piles. Écrire les opérations suivantes : `EstVide(F)`, retourne vrai si la file est vide, faux sinon; `Enfiler(F,e)` qui insère e en queue de la file F, `Defiler(F)` retourne l' élément de tête et le supprime de la file, `Tête(F)` qui retourne l' élément de tête sans le supprimer de la file. Pour toute pile P vous avez à disposition les opérations de base suivantes pour la manipuler : `EstVide(P)`, `Empiler(P,e)`, `Depiler(P)` et `Sommet(P)`.

Exercice 4

On veut manipuler des listes. Écrivez les fonctions suivantes de manière récursive en utilisant les primitives `tête(L)`, `queue(L)` et `estVide(L)`. (on suppose que toutes les listes contiennent des entiers)

1. paire(L) qui retourne *vrai* si la liste L ne contient que des entiers pairs, $faux$ sinon.
2. avant_dernier(L) qui renvoie l'avant-dernier élément de la liste L s'il existe, $faux$ sinon.