Algorithmique et structures de données
TD3 – Manipulation de listes chaînées

Lamia BENAMARA

Rappel : structure d'un elément d'une liste

Structure type **Element**
    info, entier, *l'info de l'élément*
    succ, ref Element, *l'element suivant*
FinType

Le type **Liste** est simplement une **référence** sur un élément
LSC : Liste simplement chaînée
LDC : Liste doublement chaînée

I. Niveau introductif :

Exercice 1 : Nombre d'occurrences
Concevoir un algorithme (itératif et récursif) qui prend en paramètre une liste et un élément e et retourne le nombre d'occurrences de e dans la liste

Exercice 2 : Position d'un élément
Concevoir deux algorithmes, l'un itératif, l'autre récursif, qui retournent la position d'un élément donné en argument dans une liste. Ils doivent renvoyer -1 dans le cas où l'élément ne se trouve pas dans la liste.

Exercice 3 : Valeur maximale
Ecrire un algorithme itératif (et récursif) qui permet de déterminer la valeur maximale d'une liste chaînée d'entiers positifs.

Exercice 4 : Afficher certains éléments d'une liste
Concevoir un algorithme qui affiche un élément sur deux d'une LSC.
Exemple :
Si la liste de départ est 5-> 12-> 42-> 55-> 42, votre algorithme devra afficher 5-> 42-> 42.

Exercice 5 : Parcourir une liste à rebours
Concevoir un algorithme (itératif ou récursif, à votre convenance) qui affiche les valeurs d'une LSC en partant du dernier élément et en remontant au premier

Exercice 6 : Déterminer si une liste est strictement croissante
Concevoir un algorithme qui renvoie vrai lorsque les valeurs des éléments d'une LSC d'entiers passée en argument sont strictement croissantes et qui renvoie faux dans le cas contraire.

Exercice 7 : Parcourir une LSC circulaire
Concevoir deux algorithmes, l'un itératif, l'autre récursif, qui retournent la longueur d'une LSC circulaire.

Exercice 8 : Supprimer une liste
Écrire un algorithme qui détruit une liste simplement chainée linéaire en libérant tous ses maillons. La iste est vide après exécution de l'algorithme.

II. Niveau intermédiaire :

Exercice 1 : Concaténer deux listes
On considère 2 listes chaînées l1 et l2 dont les éléments sont des entiers. Ecrire un algorithme qui rattache la liste l2 à la suite de la liste l1. Tous les cas particuliers doivent être pris en compte.

Exercice 2 : Ajouter un élément à une liste
Concevoir un algorithme qui ajoute un élément donné en argument à une position donnée dans une LSC. Si cette position est supérieure à la longueur de la liste, ajouter l'élément en queue de liste. Si la position donnée est 0, ajouter l'élément en tête de liste.
Adapter ensuite cet algorithme pour faire la même chose avec une LDC.

Exercice 3 : Supprimer un élément d'une liste
Concevoir deux algorithmes, l'un itératif, l'autre récursif, qui suppriment toutes les occurrences d'un élément donné en argument dans une LSC.
Adapter ensuite ces deux algorithmes pour faire la même chose avec une LDC.

Exercice 4 : Séparer une liste en deux
Soit une liste L de nombres relatifs.
Ecrire un algorithme permettant d'éclater cette liste en deux listes : L1 ne comportant que des entiers positifs ou nuls, et L2 des nombres négatifs.

Exercice 5 : Inverser une liste
Concevoir deux algorithmes, l'un itératif, l'autre récursif, qui inversent l'ordre d'une LSC. Il ne faut pas créer une nouvelle liste, il ne sera donc pas nécessaire d'effectuer d'allocation ou de libération de mémoire.
Adapter ensuite ces deux algorithmes pour faire la même chose avec une LDC.

Exercice 6 : Fusion de deux listes
Concevoir un algorithme qui prend en argument deux LSC dont les valeurs des éléments sont croissantes et qui retourne une seule LSC avec ces éléments dans un ordre croissant. Il ne faut pas créer une nouvelle liste, il ne sera donc pas nécessaire d'effectuer d'allocations ou de libérations de mémoire.
Exemple Soit deux LSC : a) 2 -> 7 -> 9 -> 12 -> 13 ->x et b) 1 -> 2 -> 4 -> 5 -> 15 -> x.
L'algorithme qui recevrait en argument ces deux listes renverrait alors
1 -> 2 -> 2 -> 4 ->5 -> 7 -> 9 -> 12 -> 13 -> 15 -> x.