Université
Caen Normandie

Année 2018-2019

Portail Mathématiques, Informatique
Licence deuxième année (L2 Info, L2 maths, L2 MIASHS)

Algorithmique et structures de données
TD 2 – les listes chaînées

On considère des listes d'entiers représentées par une structure simplement chaînée. On définit la structure *nœud* suivante :

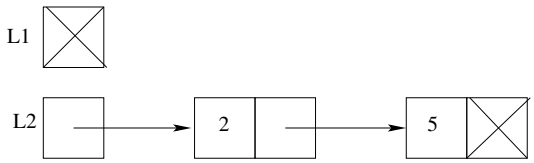
Structure nœud {
    valeur : entier
    suivant : pointeur sur nœud
}

Une liste est identifiée par un pointeur sur le premier nœud.
La liste vide est le pointeur None.

On rappelle que l'on définit le type liste avec l'instruction

type liste = pointeur sur nœud

Question 1. Écrivez les instructions nécessaires pour obtenir les listes $L1$ et $L2$ suivantes



On définit la procédure *creerListe* suivante

1 *creerListe*(n : entier) : pointeur sur nœud

```
nouvelleListe : pointeur sur nœud
nouvelleListe = None
tmp : pointeur sur nœud
pour i de n à 1 par pas de -1 faire
    Nouveau(temp)
    tmp→valeur = i
    tmp→suivant = nouvelleListe
    nouvelleListe = tmp
retourner nouvelleListe
```

Question 2. Détaillez l'exécution pas à pas du programme suivant

L : liste ; L = creerListe(2)

Question 3. Dessinez la liste L après la deuxième et la cinquième instruction du programme suivant :

1: L = pointeur sur nœud
2: L = creerListe(3)
3: P : pointeur sur nœud
4: P = L
5: P→suivant→val = 8

Question 4. Que fait la procédure *mystere* suivante ? Exprimez en fonction de la longueur de la liste le nombre d'opérations élémentaires effectuées. Donnez une version récursive de *mystere*.

---

2 *mystere*(liste : pointeur sur nœud) : entier

```
compteur : entier
compteur = 0
tant que liste<>None faire
    compteur = compteur + 1
    liste = liste→suivant
retourner compteur
```

Question 5. Écrivez une procédure récursive *affichePairsRec* qui en argument en entrée une liste et qui affiche dans le même ordre les éléments de la liste qui sont de valeur paire (les éléments de valeur impaire ne sont pas affichés). On définit comme coût le nombre de nœuds visités. Quel est le coût de *affiche_pairs* ?

Question 6. Écrivez *affichePairsIt* la même procédure mais sous forme itérative.

Question 7. Écrivez une procédure récursive *affichePairsInverseRec* qui prend en argument une liste et qui affiche dans l'ordre inversé les éléments de la liste qui sont de valeur paire (les éléments de valeur impaire ne sont pas affichés).

Question 8. * Écrivez une procédure itérative *ajoutEnFinIt* qui prend en argument une liste L et un entier x et qui retourne la liste L obtenue après insertion de x en fin. Donnez une version récursive de *ajoutEnFinIt*.

Question 9. Écrivez une procédure *testCroissant(liste : pointeur sur nœud) : booleen* qui retourne VRAI si les valeurs des nœuds de la liste sont rangés par ordre croissant et FAUX sinon. Par convention, nous retournerons VRAI si la liste est vide ou constituée d'un seul nœud.

Question 10. * Écriver une procédure récursive *copiePairs(L : liste) : liste* qui prend en argument une liste L et retourne une nouvelle liste formée des éléments de L de valeur paire, dans le même ordre. On ne modifiera pas la liste L.
Indication : pour cela, on distinguera les trois cas suivants :

1. la liste est vide
2. le premier élément de la liste est un entier impair
3. le premier élément de la liste est un entier pair.