Polytech'Paris-Sud 2015-2016
Apprentissage informatique 3ème année

TP n° 4 Bases du développement logiciel : listes chaînées

1-Introduction:
Pour ce TP, un squelette de programme est disponible à l'adresse `www.lri.fr/~khabou/BasesDevLogiciel/TP4`.

Le nom des fonctions de test est le nom de la fonction implantant un algorithme préfixée par *test_* .

Les listes chaînées manipulées sont composées de noeuds (type *node*) composé d'une valeur entière *data* et d'un pointeur *next* vers le noeud suivant. Trois fonctions sont fournies:

- node* *node_constructor*(int x) qui construit un noeud avec $data = x$ et $next = NULL$,
- node* *node_copy*(node *n) qui construit un noeud à partir d'un noeud (constructeur de copie),
- void *node_destructor*(node *n) qui détruit (désalloue) le noeud pointé.

Dans la suite, tout noeud isolé (en dehors d'une liste) doit obligatoirement avoir son pointeur *next* à *NULL*. Cela est nécessaire car la fonction *node_constructor(node, name)* teste ce pointeur pour savoir s'il faut afficher ou pas le nom *name* du noeud et pour ajouter (ou pas) un retour chariot. Regarder le code de la fonction *node_display*. Une fonction d'erreur est aussi fournie *node_error()*. Elle peut servir pour renvoyer des messages d'erreur durant la mise au point des algorithmes. Elle doit surtout servir à afficher deux erreurs très importantes pour empêcher qu'un comportement erroné ait lieu:

- *NODE_EMPTY* lors qu'on veut accéder à un noeud vide c'est à dire avec un pointeur non alloué égal à *NULL*,
- *LISTE_EMPTY* lors qu'on veut accéder à une liste vide, c'est à dire avec un pointeur non alloué égal à *NULL*,
- *LISTE_TOO_SHORT* lors qu'on veut accéder (pour insérer ou supprimer) un noeud au delà de la fin de la liste chainée.

Un exemple d'appel à la fonction d'erreur est donné dans la fonction principale de test. Le mettre en commentaire une fois analysé.

Afin de simplifier le fonctionnement et la mise au point des algorithmes, tout pointeur doit être systématiquement initialisé à *NULL* lors de sa

déclaration.

2-Fonctions de base:

1. Coder la fonction *node_ empty* qui renvoie 1 si le pointeur est *NULL* et 0 sinon.
2. Coder la fonction *liste_ len* qui renvoie la longueur de la chaine et 0 si la liste est vide.
3. Coder la fonction *liste_ display*(node *liste, char *name) qui appelle *node_ display_ all* (qui appelle en séquence la fonction *node_ display*(node *liste, char *name)). Lors du premier appel, le nom (*name*) devra être affiché, puis les différents éléments seront affichés sur la même ligne (via la chaine de formatage "%"). À la fin un retour à la ligne est normalement inséré par la fonction *node_ display* car le dernier élément doit pointer ver *NULL*. Fonction à tester après avoir réalisé *liste_ insert_ tail*.
4. Coder la fonction *liste_ destructor* qui appelle *node_ destructor* en séquence. Fonction à tester après avoir réalisé *liste_ insert_ tail*.

3-Fonctions d'insertion:

Dans cette partie et la suivante, l'hypothèse faite est que la numérotation d'une liste commence à 0: une liste de n éléments a donc des éléments sur l'intervalle $[0; n-1]$.

1. Coder la fonction *liste_ insert_ tail* qui insère un noeud en fin de liste.
2. Coder la fonction *liste_ insert_ head* qui insère un noeud en tête de liste.
3. Coder la fonction *liste_ insert* qui insère un noeud dans la liste, c'est à dire pour l'intervalle d'indices $[0; n-1]$. Cette fonction génère une erreur *LISTE_ TOO_ SHORT* si on souhaite insérer un élément au-delà de la fin de la liste ($\geqslant n$).

Des exemples d'insertion sont donnés figure 1.

4-Fonctions d'extraction:

1. Coder la fonction *liste_ extract_ tail* qui extrait le noeud de début de liste.
2. Coder la fonction *liste_ extract_ head* qui extrait le noeud de fin de liste.

3. Coder la fonction liste_extract qui extrait un noeud de la liste, c'est à dire pour l'intervalle d'indices $[0; n-1]$. Cette fonction génère une erreur *LISTE_TOO_SHORT* si on souhaite extraire un élément au-delà de la fin de la liste ($\geqslant n$).

Des exemples d'extraction sont donnés figure 2.

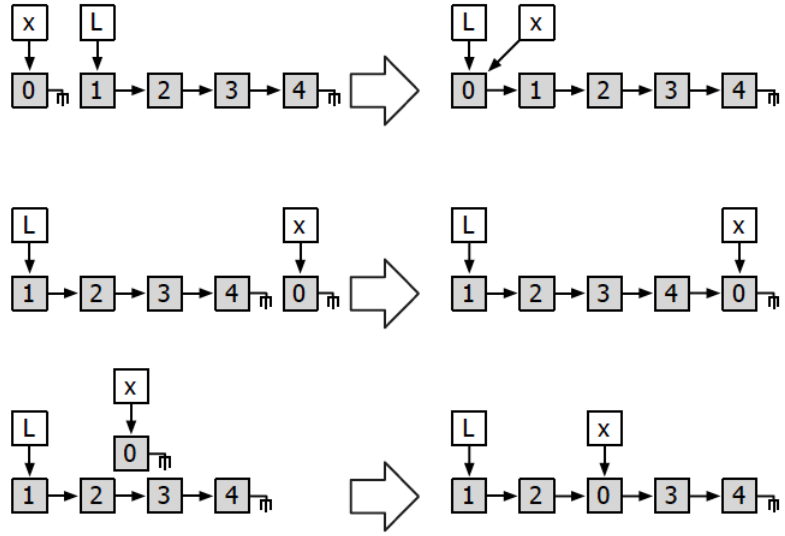

Figure 1: insertion en début (0), insert après le numéro 2, insertion en fin (3)

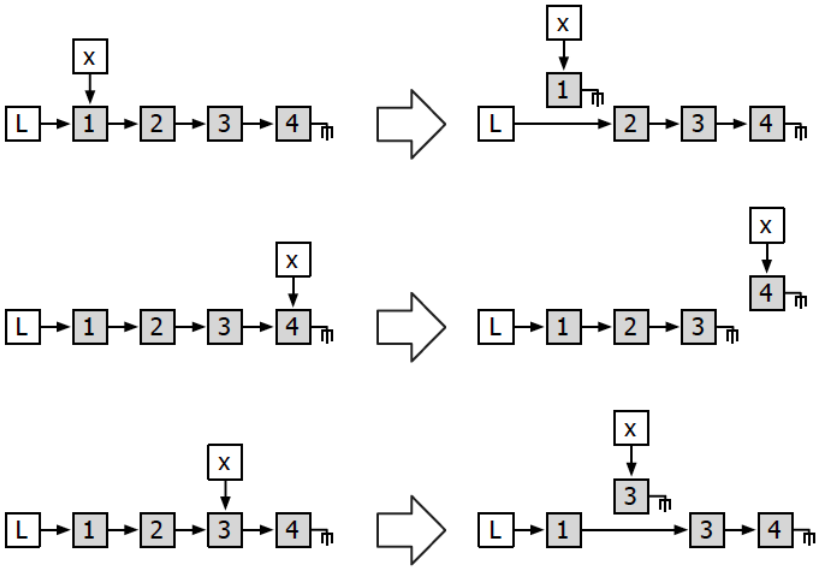

Figure 2: extraction en début (0), extraction en 2, extraction en fin (3)