TD2 - Liste, File, Pile

Exercice 1 – Liste doublement chaînée

Un des avantages des listes chaînées est de pouvoir allouer de la mémoire aux moments utiles, et de la libérer lors de la destruction d'un élément. Pour permettre un accès facile à l'élément fin de la liste ou pour accéder facilement à l'élément précédent un élément, on peut considérer une liste doublement chaînée (voir figure 1). Chaque élément pointe sur l'élément qui le précède (s'il y en a un) et sur l'élément qui le suit (s'il y en a un). Le pointeur `premier` permet de parcourir la liste en partant du premier élément et le pointeur `dernier` en partant du dernier élément.

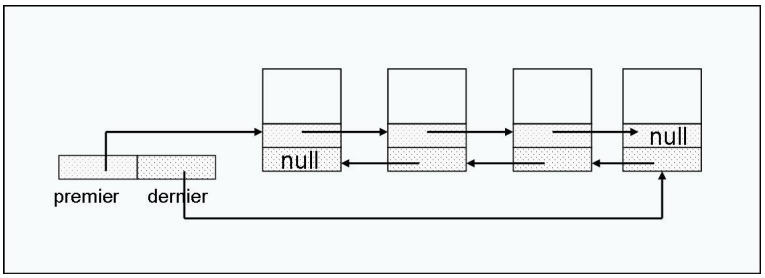


FIGURE 1 – Liste doublement chaînée

Remarque : Dans cet exercice, les valeurs stockées dans la liste sont des entiers. Toutefois, les fonctions programmées sont indépendantes du type de la valeur rangée dans la structure de liste (il suffirait de remplacer le `int` par un autre type pour disposer d'un même ensemble de fonctions).

Q 1.1 Définir la structure d'une liste doublement chaînée.

Q 1.2 Écrire la fonction `creerElement` qui crée un élément de la liste. La valeur qui doit être contenue dans l'élément est passée en paramètre.

Q 1.3 Écrire :
- une fonction `initialiserListe` qui initialise une liste déjà allouée.
- une fonction `creerListe` qui alloue, initialise et retourne une liste.
- une fonction `listeVide` qui retourne vrai si la liste est vide, et faux dans le cas contraire.

Q 1.4 Écrire la fonction `insererEnTete` qui insère un nouvel élément en tête de liste.

Q 1.5 Écrire la fonction `insererEnFin` qui insère un nouvel élément en fin de liste.

Q 1.6 Écrire la fonction `afficher` qui affiche une liste à l'envers.

Q 1.7 Écrire la fonction `rechercher` qui recherche si une valeur est présente dans la liste et retourne un pointeur sur l'élément correspondant s'il existe, et NULL s'il n'est pas trouvé.

Q 1.8 Ecrire la fonction `supprimerElement` qui enlève d'une liste un l'élément pour lequel on passe un pointeur en paramètre : on considère que cette fonction ne sera utilisée qu'avec le pré-requis que le pointeur sur élément n'est pas NULL et qu'il pointe sur un élément de la liste.

Q 1.9 Écrire les fonctions `supprimerTete` et `supprimerFin` qui permettent de supprimer le début et la fin d'une liste respectivement, et de retourner leur valeur.

Q 1.10 Écrire la fonction `desalloueListe` qui libère toute la mémoire utilisée pour la liste.

Q 1.11 Écrire une fonction main qui utilisent des opérations définies dans les questions précédentes.

Exercice 2 – Gestion de guichets

Un bureau de poste a plusieurs guichets. À l'entrée, vous donnez votre numéro et l'opération que vous voulez réaliser. Le préposé vous enregistre au guichet correspondant. Chaque guichet correspond ainsi à une file d'attente. Ainsi, on aura pour chaque guichet une liste doublement chainée dans laquelle un nouvel arrivant est ajouté en tête et la personne qui sera appelée au guichet est en fin de liste.

Q 2.1 Quelle structure de données proposez-vous pour représenter les guichets sachant que ceux-ci sont numérotés de `0` à `n`.

Q 2.2 Écrire la fonction `creerBureauPoste` qui crée un bureau de poste à n guichets.

Q 2.3 Écrire la fonction afficherPoste qui affiche pour chaque guichet la liste d'attente.

Q 2.4 Écrire la fonction `ajouterAuGuichet` qui prend un bureau de poste, un numéro du guichet et le numéro d'identification d'une personne, et qui ajoute la personne dans la file d'attente du guichet.

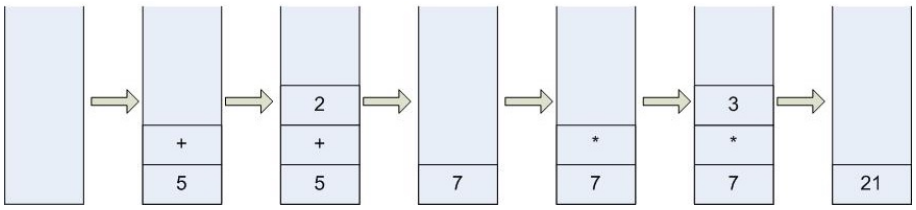
Q 2.5 Écrire la fonction `appelerAuGuichet` qui rend le numéro de la première personne appelée à ce guichet et enlève cette personne de la liste correspondante.

Exercice 3 – Évaluation d'expressions complètement parenthésées

On propose d'écrire un algorithme permettant d'évaluer une expression mathématique complètement parenthésée ne contenant que des opérateurs binaires. Pour ce faire, on utilisera une pile, appelée pile d'évaluation, qui contiendra des valeurs ou des opérateurs. L'algorithme consiste à parcourir de gauche à droite l'expression parenthésée et à empiler ou dépiler des éléments suivant le type de symbole rencontré. Pour chaque symbole de l'expression on exécute les instructions suivantes :

- Si le symbole est une valeur, on empile cette valeur.
- Si le symbole est un opérateur, on empile cet opérateur.
- Si le symbole est une parenthèse fermante, on dépile les 3 premiers éléments de la pile (un opérateur et deux valeurs) et on empile le résultat de l'opération.

La figure 2 décrit l'évolution de la pile au cours de l'évaluation de l'expression $((5 + 2) * 3)$.



Q 3.1 Écrivez l'algorithme qui permet d'évaluer une expression complètement parenthésée.