Semaine 2 – Listes simplement chainées

Ce TP va vous demander d'utiliser les fonctions d'allocation et de libération de la mémoire. Afin de simplifier le code à écrire, il n'est pas demandé de vérifier que les allocations se sont bien passées. Un programme réel devrait faire ces vérifications !

1 Mise en place

Dans ce TP, nous cherchons à implémenter un ensemble de structures de données et de fonctions permettant d'utiliser des listes simplement chainées d'entiers. Chaque noeud de la liste doit donc contenir une valeur entière ainsi qu'un pointeur vers le noeud suivant.

Commençons par les éléments de base nécessaires à la création d'une liste et à son affichage afin de simplifier les tests des fonctions suivantes. N'oubliez pas de bien tester toutes vos fonctions au fur et à mesure que vous les écrivez et n'oubliez pas de tester les cas limites.

1. Commencer par définir les deux structures nécessaire à la gestion d'une liste chainée : une structure pour les noeuds et une autre pour la liste elle même. On suppose que la structure de liste garde une trace du nombre d'éléments afin de ne pas avoir à parcourir la chaine si celui-ci est nécessaire.
2. Écrivez maintenant une fonction permettant de créer une liste vide. Attention de bien initialiser tout ce qui doit l'être.

   ```
   liste *nouvelle_liste();
   ```
3. Et enfin, une dernière fonction basique va nous être utile : une fonction permettant d'afficher le contenu d'une liste :

   ```
   void afficher_liste(liste *l);
   ```

2 Fonction d'édition

Il s'aggit maintenant d'implémenter les fonctions de base permettant de modifier une liste, c'est-à-dire ajouter et supprimmer des éléments. De nombreuses fonctions de ce type sont possibles et chacune à son utilitée, nous nous contenterons ici des plus simples permettant de manipuler un minimum nos listes.

1. Écrivez une fonction permettant d'ajouter un élément donné en paramètre en tête d'une liste. L'ajout en tête est souvent le type d'ajout le plus simple et le plus rapide pour les listes simplement chainées.

   ```
   void ajouter_en_tete(liste *l, int val);
   ```
2. Ajouter maintenant une fonction permettant de chercher la première occurence d'une valeur dans une liste et de la supprimer. Cette fonction de suppression n'est pas la plus courante pour les listes chainée mais elle nous sera bien utile plus tard et se combine bien avec la fonction que nous allons écrire à la prochaine question.

   Attention : les choses se compliquent avec cette fonction, pensez à bien la tester, les cas limites sont ici importants et plus nombreux.

   ```
   void supprimer_valeur(liste *l, int val);
   ```

3 Fonctions de recherche

Les fonctions permettant d'extraire des valeurs d'une liste chainée sont aussi très variées. Elles peuvent permettrent de retrouver une valeur en fonction de sa position dans la liste (en tête ou en queue par exemple) ou en fonction de sa valeur (minimum ou maximum) par exemple.

Dans notre cas, nous nous contenterons d'une seule fonction mais n'hésitez pas à en implémenter d'autre pour vous entrainer.

1. Écrivez une fonction permettant de retrouver la plus grande valeur stockée dans une liste donnée :

   ```
   int chercher_maximum(liste *l);
   ```

4 Approfondissements

Toutes les fonctions implémentéeis dans les question précédentes sont suffisantes pour la suite des TP et vous êtes encouragés à vous assurer qu'elles fonctionnent parfaitement avant de poursuivre.

1. Nous ne disposont actuellement d'aucun moyen de libérer la mémoire occupée par une liste. Ajoutez une fonction permettant de libérer toute la mémoire qui a été allouée pour les noeuds de la liste ainsi que pour la liste elle même.

   Attention : n'oubliez pas que vous ne devez jamais accéder au contenu d'une structure après en avoir libérer la mémoire.

   ```
   void liberer_liste(liste *l);
   ```

2. Afin de diversifier les possibilités d'utilisation de notre liste, ajoutez une fonction permettant d'insérer une valeur en queue de liste cette fois-ci :

   ```
   void ajouter_en_queue(liste *l, int val);
   ```