TP6 : Listes chaînées unilatères

Séances 10 et 11 de TP de C – Evaluation du binôme en séance 11

1. Objectifs

L'objectif de cette séance est d'appliquer, en langage C, les notions d'algorithmiques sur les listes chaînées unilatères. Vous aurez donc à coder une bibliothèque permettant la création et la manipulation de listes chaînées.

Les notions de langages C qui vous seront utiles sont les structures (attention à l'utilisation des « typedef »), les pointeurs, ainsi que l'allocation dynamique en mémoire.

Vous développerez cette bibliothèque dans des fichiers séparés afin de rendre votre code réutilisable. Vous devrez mettre en place des tests unitaires (toujours avec le module « seatest ») qui permettront de vérifier le bon comportement de vos fonctions de base.

L'utilisation de la fonction « scanf » est de nouveau interdite, même durant la phase de développement et de mise au point. Vous devrez donc concevoir vos tests unitaires en même temps que vos fonctions.

2. Les fonctionnalités attendues

Ci-dessous est listé l'ensemble des fonctionnalités que devra fournir votre bibliothèque. Chacune de ces fonctionnalités possédera une fonction de test unitaire.

Le temps de codage est indiqué à titre informatif. Il correspond au temps qu'il devrait vous falloir pour coder chaque série de fonctionnalités, actuellement. Si vous mettez plus de temps, c'est que vous devez vous entraîner et pratiquer chez vous pour

augmenter votre vitesse et ne plus être bloqué par des points de cours.

Les fonctions basiques

Tout cela a déjà été codé en TD d'algorithmique, ça devrait aller très vite.

- Créer une liste
- Supprimer une liste
- Insérer un élément avant le -nième noeud
- Supprimer le -nième noeud
- Récupérer l'élément se trouvant au -nième noeud
- Récupérer le nombre d'élément de la liste
- Afficher la liste dans la console

Ne pas oublier les tests unitaires

 30 minutes pour les fonctions + 30 minutes pour les tests unitaires

Les fonctions d'aides à la manipulation

Ces fonctions réutilisent les fonctions de bases, ça devrait aller aussi très vite.

- Insérer un élément en début de liste
- Insérer un élément en fin de liste
- Supprimer un élément en début de liste
- Supprimer un élément en fin de liste
- Vider la liste
- Récupérer l'élément en début de liste
- Récupérer l'élément en fin de liste

Ne pas oublier les tests unitaires

 20 minutes pour les fonctions + 30 minutes pour les tests unitaires.

Les fonctions avancées

Le TP de C commence vraiment à partir d'ici : Ces fonctions n'ont pas été vues en TD.

- Échange deux noeuds de la liste
- Remplacer l'élément se trouvant au -nième noeud
- Concaténer deux listes dans une nouvelle
- Retourne la position de la première occurrence d'un élément dans la liste

 30 minutes pour les fonctions + 20 minutes pour les tests unitaires

Les fonctions bonus

Pour ceux qui ont fini de coder toutes les fonctionnalités précédentes, on va commencer à s'amuser un peu.

- Retourner une nouvelle liste, clone de la première, mais sans les doublons
- Retourner une nouvelle liste représentant l'union de deux listes
- Retourner une nouvelle liste représentant l'intersection de deux listes
- Retourner une nouvelle liste composée des éléments de la première liste qui ont passé une fonction de test. (il faut utiliser des pointeurs de fonction)

 45 minutes pour les fonctions + 25 minutes pour les tests unitaires

3. Astuces

- Démarrez un nouveau projet – Ne repartez pas d'un ancien TP – Réfléchissez à l'organisation de vos fichiers sources
- Une liste est une structure contenant un pointeur vers le premier élément de la liste, un pointeur vers le dernier élément de la liste, ainsi que la taille de la liste
- Un noeud de la liste est une structure contenant un pointeur vers le noeud suivant ainsi que l'élément en question
- Aidez-vous de vos TD d'algorithmique
- Réutilisez ce que vous avez fait au TP2 sur les tests unitaires
- Concevez vos tests unitaires avant de coder vos fonctions correspondantes

4. Evaluations

- Maîtrise de CodeBlocks (Création d‘un projet, ajout de fichiers sources c,h, compilation, debugger)
- Conventions de codage (commentaires, indentation, nom des fichiers, des fonctions, des variables, ...)
- Compilation séparée .h/.c, séparation du code en fonctions
- Tableaux, boucles
- Allocation dynamique de mémoire
- Lecture/écriture de fichiers
- Structures
- Pointeurs, passage de paramètres par valeur/par adresse
- Respect des contraintes de l'énoncé
- Tests unitaires & jeux d’essais