Colle n°5

Tous vos fichiers sont à **renommer** comme d'habitude (en y ajoutant votre nom en majuscule) et à **déposer sur Moodle** dans la section `Informatique - Devoirs`, tout en bas dans 2ème session.

Vous devrez terminer l'exercice qui vous a été attribué et me remettre l'ensemble des fichiers corrigés sur Moodle au même endroit avant dimanche soir.

Pensez à toutes les bonnes pratiques de programmation : réfléchir sur papier, tester dès que cela est possible, méthode des petits pas, programmation défensive...

Si vous avez terminé en moins d'1h, venez me demander un exercice supplémentaire.

Vous n'êtes pas autorisé à réutiliser le travail effectué en TP mardi, vous devez être capable de reconstruire de zéro. Toutefois, si vous êtes bloqué, appelez moi pour pouvoir continuer à travailler.

Exercice 1 (Listes doublement chaînées (ROBIN, ROGER)).

Une liste doublement chaînée (LDC) est une liste implémentée sous la forme de maillons chaînés qui peuvent se parcourir dans les deux sens : chaque maillon est donc relié au maillon suivant mais aussi au maillon précédent. **On considérera ici uniquement des listes de nombres réels.**

Toutes les fonctions de cet exercice seront écrites en C dans un fichier `NOM_ldc.c`

1. Proposer des structures de données en C qui implémentent concrètement ce type de listes
2. Implémenter une fonction `ldc_create` permettant de créer un objet de ce nouveau type
3. Implémenter une fonction `ldc_push` permettant d'ajouter un élément **en tête** de votre LDC
4. Implémenter une fonction `ldc_print` qui affiche tous les éléments de votre LDC
5. Tester et déboguer ces premières fonctions, en faisant des manipulations « en dur » dans la fonction principale
6. Implémenter et tester une fonction `ldc_free` qui libère tout l'espace mémoire qui a été alloué pour le stockage de la LDC.
7. Implémenter une fonction `ldc_pop` qui retourne la valeur stockée en tête de liste et supprime l'élément de tête de liste.
8. Tester abondamment cette fonction.
9. Implémenter une fonction `ldc_insert_ith` qui insère une nouvelle valeur en i-ème position (on commence la numérotation des éléments à 0)
10. Implémenter une fonction `ldc_delete_ith` qui supprime le ième élément de la liste
11. Implémenter une fonction `ldc_reverse` qui renverse l'ordre des éléments d'une liste doublement chaînée. La fonction doit travailler **en place**, c'est-à-dire sans allouer de nouveaux maillons.
12. Facultatif : Implémenter une fonction `ldc_tri_insertion` qui trie une liste doublement chaînée par valeurs croissantes avec l'algorithme de tri par insertion.

Exercice 2 (Implémentation des listes avec un grand tableau (PISCH, YAO)).
Dans un fichier C nommé `NOM_listes_tableau.c`, implémenter toutes les fonctions dont le prototype est donné ci-dessous, en utilisant une implémentation utilisant un grand tableau comme vu en cours. La taille du grand tableau pourra être fixée en dur dans le code, dans le constructeur.

```c
// constructeur
extern list *create_list();

// accesseurs
extern int get_length(list *l);
extern int length(list *l);
extern int get_ith(unsigned int i, list *l);


// transformateurs
extern void push(elt_type v, list *l);
extern void print_list(list *l);
extern void delete_ith(unsigned int i, list *l);
extern void insert_ith(unsigned int i, elt_type v, list *l);
extern list *concatenate(list *l1, list *l2);

// destructeur
extern void free_list(list **addr_l);
```

Tester et déboguer abondamment en faisant des manipulations « en dur » dans la fonction principale. Appliquez également tous les principes de la programmation défensive.

Exercice 3 (Base de données patients (THIRY, KLEIN)).

Toutes les fonctions de l'exercice seront codées dans un fichier `NOM_patients.c`
Privilégiez les choix d'architecture, la modularité et le découpage de votre code. Lisez l'ensemble de l'énoncé et faites le squelette avant de foncer sur votre clavier.
Vous avez beaucoup de liberté : faites tous les choix qui vous sont utiles et /ou vous simplifient la vie.
Et n'oubliez pas : méthode des petits pas.
Pensez en terme de **produit minimum viable**. Vous pourrez l'améliorer ensuite autant que vous voudrez.

Vous allez créer un code qui permet de manipuler une petite base de données de patients pour un médecin.
Un patient est inscrit dans la base de données avec les informations suivantes :

- nom (max 20 caractères),
- prénom (max 20 caractères)
- âge
- numéro de sécurité sociale (15 caractères numériques)

Votre logiciel devra permettre à la personne en charge du secrétariat du médecin de maintenir à jour une base de données de patients

- lorsque la personne en charge du secrétariat lance le logiciel en arrivant en début de journée, la base client existante, stockée dans un fichier texte nommé `patients.bdd`, est chargée en mémoire de travail dans une structure de données adaptée ;
- au fur et à mesure de la journée, la personne en charge du secrétariat rentre en ligne de commande les informations des patients se présentant au cabinet. Si un patient est déjà présent dans la base de données (même nom), par exemple parce qu'il est déjà venu chez ce médecin, le logiciel ne fait rien de plus ;
- à la fin de la journée, où simplement lorsque la personne en charge du secrétariat arrête le logiciel, une sauvegarde de la base de données patient est effectuée en réenregistrant toutes les informations dans le fichier `patients.bdd` sur le disque dur.

1. Réfléchissez aux structures de données permettant d'implémenter un tel logiciel et aux opérations nécessaires (a minima) sur ces structures de données pour que votre logiciel puisse réaliser le travail demandé. Il faudra être capable de justifier vos choix.

2. Créer les structures de données et le squelette de votre code, avec toutes les fonctions à implémenter, en prenant soin de choisir des noms parlants pour vos fonctions et leurs paramètres.

3. Nous allons temporairement oublier la problématique de sauvegarde de la base de données dans un fichier. On imagine qu'à chaque lancement du logiciel, on part d'une base de données patients vierge. On oublie également pour le moment la problématique de la sauvegarde de la base de données dans un fichier lors de l'arrêt du logiciel.

a. Implémenter prioritairement la fonction qui permet à la personne du secrétariat d'enregistrer un patient qui se présente au cabinet, sans vous préoccuper de la présence ou non de ce patient dans la base.

b. Implémenter une fonction permettant d'afficher l'état courant de la base de données en mémoire

c. Tester ces premières fonctions.

4. Nous allons maintenant nous intéresser au chargement et à la sauvegarde de la base de données, qui est stockée dans un fichier texte.

a. Réfléchissez à un format pour le fichier de sauvegarde de votre base de données client. Faites simple.

b. Coder la fonction permettant de charger la base de données existante

c. Coder la fonction permettant de sauvegarder la base de donnée à la fin de la journée, lorsque la personne du secrétariat éteindra le logiciel.

d. Tester !

5. Que pensez vous de ce logiciel et de sa stratégie de gestion des données ?