TP n° 5

I3 : Structures de données
Lecture et écriture dans des fichiers

Exercice 1 [Les étudiants de l'IUT]

Vous avez vu dans le TD n°5 la conception d'une bibliothèque de gestion des étudiants de l'IUT.

1. Implémentez les fonctions creerEtudiant(), copierEtudiant(), libererEtudiant() et afficheEtudiant() dans la bibliothèque libmanipEtudiant.so, en utilisant l'outil make pour la compilation, comme vu dans le TD n°5.
2. Écrivez un programme de test qui fait appel aux fonctions de la bibliothèque libmanipEtudiant.so
3. Dans la bibliothèque, écrivez une procédure qui remplit un tableau d'étudiants (une dizaine d'étudiants) passé en paramètre.
4. Dans la bibliothèque, écrivez une procédure en C qui sépare le tableau en deux autres tableaux : les étudiants de première année et les étudiants de deuxième année. Les pointeurs vers les trois tableaux (un tableau d'entrée et deux tableaux de sortie), le nombre total d'étudiants et un pointeur vers le nombre d'étudiants dans chaque tableau de sortie sont passés en paramètres de la procédure.
5. Dans la bibliothèque, écrivez une procédure en C qui affiche tous les étudiants contenus dans un tableau. Le tableau et le nombre d'étudiants sont passés en paramètres.
6. Dans le programme en C, créer un tableau de *pointeurs sur etudiant_t*. Remplissez-le avec des pointeurs vers des etudiant_t (par exemple, ceux du tableau que vous avez rempli plus tôt avec la fonction idoine). Prévoyez une case supplémentaire à la fin de votre tableau, qui contiendra le pointeur NULL.
7. Jusqu'à maintenant, les manipulations de tableau ont toujours nécessité de connaitre le nombre d'éléments dans un tableau. Écrivez une fonction dans la bibliothèque qui retourne le nombre d'éléments dans un tableau de pointeurs sur etudiant_t terminé par NULL. Attention : le tableau peut-être mal formé. L'utilisateur de la bibliothèque peut lui passer un tableau qui ne se termine pas par NULL. Par mesure de sécurité, vous prévoirez une limite à ne pas dépasser.

Exercice 2 [Les étudiants enchaînés]

Les fonctions de cet exercice sont à ajouter à la bibliothèque que vous avez écrite à la question précédente. Les prototypes (à ajouter dans le fichier d'en-têtes) sont indiqués à la fin de l'énnoncé.

1. Écrivez une structure de données qui représente un étudiant et contient un pointeur vers un autre élément du type de cette structure de données, de telle sortie que l'on puisse l'utiliser pour former une liste chaînée.
2. Ecrivez une fonction qui permet de remplir un élement de la liste avec les champs d'un étudiant.
3. Ecrivez une fonction qui permet de remplir un étudiant avec les champs d'un élément de la liste.
4. Écrivez une fonction qui permet de créer une liste chaînée vide.
5. Écrivez une fonction qui permet d'ajouter un étudiant au début de la liste.
6. En utilisant la fonction `strncmp()`, écrivez une fonction qui compare deux étudiants en fonction de leur nom de famille. Si ils portent le même nom de famille, on regardera leurs prénoms. Si les prénoms sont identiques, on regarde leurs dates de naissance (année puis mois puis jour). La fonction retourne un nombre positif si le premier étudiant est supérieur selon ces critères au second, négatif si le premier étudiant est inférieur au second, zéro si ils sont identiques.

7. Écrivez une fonction qui compare de la même façon deux éléments d'une liste. Vous pourrez utiliser la fonction précédente.
8. Écrivez une fonction qui enliste un étudiant dans la liste chaînée de telle sorte que les étudiants de la liste soient triés selon l'ordre défini dans les questions précédentes.
9. Écrivez une fonction qui permet de convertir un tableau de pointeurs vers des étudiants en une liste chaînée triée par noms.
10. Écrivez une fonction qui retourne le nombre d'éléments dans la liste chaînée.
11. Écrivez une fonction qui affiche tous les éléments de la liste chaînée.
12. Écrivez une fonction qui cherche un étudiant dans la liste. Cette fonction prend en paramètres son nom, son prénom (chaînes de caractères) et un pointeur vers la liste. Elle retourne un pointeur vers cet élément. Si l'étudiant n'a pas été trouvé, la fonction retourne NULL.
13. Écrivez une fonction qui supprime un élément de la liste. Un pointeur vers cet élément est passé en paramètre. La fonction retourne NULL si l'élément n'a pas été trouvé. Pourquoi doit-on retourner le pointeur vers le début de liste ?
14. Écrivez une fonction qui surrpime un par un tous les éléments d'une liste.
15. Écrivez une fonction qui supprime un étudiant de la liste chaînée. Cette fonction prend en paramètre le nom de famille et le prenom de l'étudiant (chaîne de caractères) et un pointeur vers la liste chaînée et retourne un pointeur vers la liste chaînée ou NULL si l'étudiant n'a pas été trouvé.
16. Écrivez une fonction qui permet de convertir une liste chaînée en un tableau d'étudiants.

Les fonctions et procédures à ajouter dans la biliothèque ont les prototypes suivants ;

```
1  struct liste_etudiant* nouvelle_liste();
2  struct liste_etudiant* ajouter_debut( struct liste_etudiant* liste,
3                                        etudiant_t et);
4  int comparerEtudiants( etudiant_t et1, etudiant_t et2);
5  struct liste_etudiant* ajouter( struct liste_etudiant* liste, etudiant_t et );
6  struct liste_etudiant* tab2chain( etudiant_t** tableau );
7  etudiant_t* chain2tab( struct liste_etudiant* liste );
8  etudiant_t elem2etu( struct liste_etudiant* elem );
9  struct liste_etudiant* etu2elem( etudiant_t et );
10 int nbElem( struct liste_etudiant* liste );
11 void afficheListe( struct liste_etudiant* liste );
12 struct liste_etudiant* chercheEtud( struct liste_etudiant* liste,
13                                     char* nom, char* prenom );
14 struct liste_etudiant* suprElem( struct liste_etudiant* liste,
15                                  struct liste_etudiant* elem );
16 struct liste_etudiant* suprEtudiant( struct liste_etudiant* liste,
17                                      char* nom, char* prenom );
18 void effacerListe( struct liste_etudiant* liste );
```

Exercice 3 [Les étudiants persistents]

On souhaite conserver la liste des étudiants dans un fichier afin de pouvoir récupérer les données même après avoir quitté l'application. Pour cela, nous allons écrire ces données dans un fichier, et les charger depuis un fichier.

Cet exercice fait suite aux deux précédents. Vous ajouterez les fonctions écrites ici dans la bibliothèque que vous avez écrite dans les exercices précédents.

1. Écrivez une fonction qui prend en paramètre un nom de fichier (chaîne de caractères) et un pointeur vers une liste chaînée et écrit dans le fichier, en mode caractères, les informations sur les étudiants. Chaque étudiant est écrit sur une ligne. Les informations sont séparées par des tabulations (caractère \t). La fonction retourne le nombre d'octets écrits (-1 en cas de problème). N'oubliez pas d'ouvrir et de fermer votre fichier... Si le fichier existe déjà, il est écrasé.
2. Écrivez une fonction qui charge les données depuis un fichier. Les données sont contenues dans le fichier au format décrit dans la question précédente. La fonction prend en paramètre un nom de fichier (chaîne de caractères) et retourne un pointeur vers le début d'une liste chaînée contenant les données des étudiants, ou NULL si un problème a été rencontré. La liste doit être triée.

Les fonctions et procédures à ajouter dans la biliothèque ont les prototypes suivants ;

```
1 int ecrireListe( char* nomFichier, struct liste_etudiant* liste );
2 struct liste_etudiant* lireListe( char* nomFichier );
```