Mini-Projet 2 : Gestion d'une bibliothèque

Ce mini-projet s'étale sur deux séances de TME (séances 3 et 4). Dans ce mini-projet, nous nous intéressons à la gestion d'une bibliothèque où une bibliothèque est un ensemble de livres. Un livre est repéré par son titre, le nom de son auteur et un numéro d'enregistrement. Plus précisément, un livre est représenté par les données suivantes :

```
1   int num;
2   char *titre;
3   char *auteur;
```

L'objectif de ce mini-projet est d'apprendre à comparer des structures de données. Plus précisément, nous utiliserons pour implémenter une bibliothèque :

- Une liste simplement chaînée de `struct` (Partie 1).
- Une table de hachage de `struct` (Partie 2).

Vous travaillerez avec la bibliothèque stockée dans le fichier nommé `GdeBiblio.txt` (à télécharger depuis moodle), dont le format est très simple :

- Chaque livre (une entrée) correspond à une ligne.
- Chaque ligne comprend le numéro du livre, suivi de son titre et de son auteur, séparés par un espace.

Pour simplifier la lecture et l'écriture de fichiers, nous supposons dans ce projet que les titres et les auteurs de livres ne contiennent pas d'espaces.

Exercice 1 – Gestion d'une bibliothèque avec une liste chaînée de struct

Dans ce premier exercice, nous allons coder une bibliothèque comme une liste chaînée de `struct` de type livre. On utilisera alors la structure suivante :

```
1  typedef struct livre{
2     int num;
3     char *titre;
4     char *auteur;
5     struct livre * suiv;
6  } Livre;
7
8  typedef struct{ /* Tete fictive */
9     Livre* L;  /* Premier element */
10 } Biblio;
```

Q 1.1 Créez un fichier "biblioLC.h" dans lequel vous ajouterez la définition de cette structure. Dans ce fichier, vous ajouterez au fur et à mesure les signatures des fonctions (opérations) que vous coderez plus tard pour cette structure.

Q 1.2 Créez un fichier "biblioLC.c" dans lequel vous ajouterez progressivement le code des fonctions manipulant cette structure. Commencer par y ajouter les fonctions suivantes :

- `Livre* creer_livre(int num,char* titre,char* auteur)` qui crée un livre.
- `void liberer_livre(Livre* l)` qui réalise une désallocation.

- `Biblio* creer_biblio()` qui crée une bibliothèque vide.
- `void liberer_biblio(Biblio* b)` qui libère la mémoire occupée par une bibliothèque.
- `void inserer_en_tete(Biblio* b,int num,char* titre,char* auteur)` qui ajoute un nouveau livre à la bibliothèque.

Q 1.3 Créez un fichier "entreeSortieLC.h" qui contiendra les signatures des fonctions permettant de manipuler des fichiers. Dans le fichier "entreeSortieLC.c", écrivez les fonctions suivantes :

- `Biblio* charger_n_entrees(char* nomfic, int n);` permettant de lire n lignes du fichier et de les stocker dans une bibliothèque.
- `void enregistrer_biblio(Biblio *b, char* nomfic);` qui permet de stocker une bibliothèque dans un fichier au bon format : numéro titre auteur.

Q 1.4 Créez un fichier "main.c" contenant une fonction `int main(int argc, char** argv)` permettant à l'utilisateur de donner directement en ligne de commande le nom du fichier contenant la bibliothèque à lire, ainsi que le nombre de lignes à lire dans le fichier. Autrement dit, avec la commande `./main GdeBiblio.txt 100`, le programme doit lire 100 lignes du fichier appelé "GdeBiblio.txt".

Rappel :

- `argv` est un tableau de chaînes de caractères qui est composé d'une chaîne de caractères par mot de la ligne de commande. Dans l'exemple, "./main" compte comme un mot, "GdeBiblio.txt" est un deuxième mot, et "100" est un troisième mot.
- `argc` est le nombre de mots de la ligne de commande. Dans l'exemple ci-dessus, on a trois mots : "./main", "GdeBiblio.txt" et "100".

Q 1.5 Créez un fichier `Makefile`.

Q 1.6 Créez les fonctions suivantes permettant de réaliser :

- l'affichage d'un livre.
- l'affichage d'une bibliothèque.
- la recherche d'un ouvrage par son numéro.
- la recherche d'un ouvrage par son titre.
- la recherche de tous les livres d'un même auteur (retourne une bibliothèque).
- la suppression d'un ouvrage (à partir de son numéro, son auteur et son titre).
- la fusion de deux bibliothèques en ajoutant la deuxième bibliothèque à la première, et en supprimant la deuxième.
- la recherche de tous les ouvrages avec plusieurs exemplaires. Deux ouvrages sont identiques s'ils ont le même auteur et le même titre (seul le numéro change). Cette fonction devra renvoyer une liste comprenant tous les exemplaires de ces ouvrages, avec une complexité-temps pire cas en $O(n^2)$ où n est la taille de la bibliothèque.

Q 1.7 Dans le fichier "main.c", écrire une fonction `void menu()` permettant d'afficher à l'utilisateur toutes les actions possibles sur la bibliothèque. Par exemple : 0-sortie du programme, 1-Affichage, 2-Inserer ouvrage, etc.

Q 1.8 Dans la fonction `main`, faire une boucle qui :

- affiche le `menu` à l'utilisateur,
- récupère l'action à réaliser en lisant sa réponse au clavier,
- puis réalise l'action souhaitée par l'utilisateur.

Cette boucle s'arrête quand l'utilisateur tape '0', indiquant la sortie du programme. On peut par exemple utiliser un `switch case` comme ceci :

```
...
/* Lecture du fichier contenant la bibliotheque (voir question 1.4) */
...
int rep;
do{
    menu();
    scanf("%d",&rep)
    switch (rep){
    case 1:
        printf("Affichage :\n");
        afficher_biblio(B);
        break;
    case 2:
        int num;
        char titre[256];
        char auteur[256];
        printf("Veuillez ecrire le numero, le titre et l'auteur de l'ouvrage.\n")
        /* On suppose que le titre et l'auteur ne contiennent pas d'espace*/
        if (scanf("%d%s%s",&num,titre,auteur)==3){
            inserer_en_tete(B,num,titre,auteur);
            printf("Ajout fait.\n")
        }else{
            printf("Erreur format\n");
        }
        break;

        etc.
    }
}while(rep!=0);
printf("Merci, et au revoir.\n")
return 0;
```

Attention : Dans le cours, nous avons vu qu'il était préférable d'utiliser la fonction `fgets` plutôt que `scanf`. Nous vous demandons donc d'utiliser la première fonction plutôt que la deuxième.

Exercice 2 – Gestion d'une bibliothèque avec une table de hachage

Bien que les listes soient très souples (allocation et libération dynamiques de la mémoire pour l'ajout et la suppression de données), elles ne sont pas très efficaces quand il s'agit d'effectuer de nombreuses recherches (comme c'est le cas avec une bibliothèque...). Pour accélérer l'accès à un élément de la bibliothèque, nous allons utiliser maintenant une table de hachage avec résolution des collisions par chaînage. Pour implémenter la table de hachage, vous pourrez utiliser les `struct` suivantes :

```
typedef struct livreh{
  int clef;
  /* int num; ... toutes les donnees permettant de representer un livre */
  struct livreh *suivant;
} LivreH;
```

```c
typedef struct table {
  int nE; /*nombre d'elements contenus dans la table de hachage */
  int m; /*taille de la table de hachage */
  LivreH** T; /*table de hachage avec resolution des collisions par chainage */
} BiblioH;
```

Q 2.1 Créez les fichiers "biblioH.c" et "biblioH.h" dans lesquels vous écrirez le code permettant de gérer une bibliothèque avec une table de hachage.

Q 2.2 À chaque livre, nous allons associer une clé qui est un entier obtenu à partir du nom de son auteur. Plus précisément, la clé associée à un livre est la somme des valeurs ASCII de chaque lettre du nom de son auteur. Créez la fonction `int fonctionClef(char* auteur);` réalisant cette opération.

Q 2.3 Créez les fonctions suivantes :
- `LivreH* creer_livre(int num,char* titre,char* auteur)` qui crée un livre.
- `void liberer_livre(LivreH* l)` qui réalise une désallocation.
- `BiblioH* creer_biblio(int m)` qui crée une bibliothèque avec une table de taille m.
- `void liberer_biblio(BiblioH* b)` qui libère la mémoire occupée par une bibliothèque.

Q 2.4 Nous allons maintenant définir la fonction de hachage à utiliser. Par définition, cette fonction permet de transformer la clé de chaque livre en une valeur entière utilisable par la table de hachage (c'est-à-dire, un entier entre 0 et m non compris). Il est important de choisir une fonction de hachage permettant d'éviter au maximum les collisions pour obtenir des opérations de recherche/suppression efficaces. Pour ce faire, on peut par exemple utiliser la fonction de hachage :

$$h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$$

où k est la clé associée à un livre et $A = \frac{\sqrt{5}-1}{2}$ (nombre d'or diminué de 1, proposé par Donald Knuth [1]). Créez la fonction `int fonctionHachage(int cle, int m);` réalisant cette opération.

Q 2.5 Créez la fonction `void inserer(BiblioH* b,int num,char* titre,char* auteur)` qui ajoute un nouveau livre à la bibliothèque. Pour insérer un ouvrage, vous devrez procéder en deux étapes :

1. Trouver dans quelle case de la table de hachage insérer le livre (grâce à sa clé et à la fonction de hachage).
2. Insérer le livre en tête de la liste chaînée correspondant à cette case.

Q 2.6 Adaptez les questions 1.3 et 1.6 à l'utilisation d'une table de hachage pour gérer la bibliothèque. Modifier votre `Makefile` et votre fichier `main.c` pour tenir compte des tables de hachage.

Exercice 3 – Comparaison des deux structures

Dans cet exercice, on veut comparer les deux structures (liste et table de hachage) par rapport au temps nécessaire pour effectuer les fonctions de recherche : recherche simple et recherche des ouvrages avec plusieurs exemplaires.

1. Né le 10 janvier 1938 dans l'état du Wisconsin aux États-Unis, Donald Knuth est un informaticien et mathématicien américain de renom et professeur émérite en informatique à l'université Stanford (États-Unis). Il est un des pionniers de l'algorithmique et a fait de nombreuses contributions dans plusieurs branches de l'informatique théorique. Il est également l'auteur de l'éditeur de texte LaTeX.

Q 3.1 Comparer les temps de calcul entre les deux structures pour réaliser la recherche d'un ouvrage par son numéro, son titre et son auteur. Pour avoir des temps de calcul significatifs, vous pourrez procéder à plusieurs recherches consécutives. Vous pourrez aussi distinguer deux cas : livre présent et livre absent. Quelle structure (liste ou table de hachage) est la plus appropriée pour chacune de ces recherches ? Justifiez votre réponse.

Q 3.2 Modifiez la taille de votre table de hachage. Comment évoluent vos temps de calcul en fonction de cette taille ?

Q 3.3 On veut déterminer les temps de recherche des ouvrages en plusieurs exemplaires en fonction de la taille de la bibliothèque et de la structure de données utilisée. Pour cela, vous adapterez votre fonction pour qu'elle puisse relancer la fonction de lecture en lisant partiellement les n premières lignes du fichier, n prenant les valeurs de 1000 à 50 000 avec un pas croissant [2]. Pour chaque valeur de n vous sauvegarderez les temps de calcul obtenus avec chacune des deux structures. Vous visualiserez ensuite par des courbes les séries de nombres obtenues.

Q 3.4 Justifiez les courbes obtenues en fonction de la complexité pire-cas attendue.

2. Si sur votre machine le temps de calcul devient trop long pour 50 000 entrées (c'est-à-dire supérieur à 5 minutes), vous pouvez diminuer le nombre maximal d'entrées.