

Mini-Projet: Gestion d'une bibliothèque

Ce mini-projet s'étale sur deux séances de TME (séances 3 et 4). Il est noté et doit être rendu sur moodle avant la séance 5 de TME. Les instructions sur la soumission de votre code et de votre rapport sont sur moodle. Dans ce fichier, vous trouverez la partie 1 de ce mini-projet. La partie 2 sera mise sur moodle prochainement.

Dans ce 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, on a trois mots.

`argc` est le nombre de mot de la ligne de commande (`./main` compte comme un mot, `"GdeBiblio.txt"` est un deuxième mot, et `"100"` est un troisième mot).

Q 1.5 Créez un fichier `Makefile` et expliquez vos choix dans votre rapport.

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 :

```
1  ...
2  /* Lecture du fichier contenant la bibliotheque (voir question 1.4) */
3  ...
4  int rep;
5  do{
6      menu();
7      scanf("%d",&rep)
8      switch (rep){
9          case 1:
10         printf(" Affichage :\n");
11         afficher_biblio(B);
12         break;
13         case 2:
14             int num;
15             char titre[256];
16             char auteur[256];
17             printf(" Veuillez ecrire le numero, le titre et l'auteur de l'ouvrage.\n")
18             /* On suppose que le titre et l'auteur ne contiennent pas d'espace*/
19             if (scanf("%d%s%s",&num,titre,auteur)==3){
20                 inserer_en_tete(B,num,titre,auteur);
21                 printf(" Ajout fait.\n")
22             }else{
23                 printf(" Erreur format\n");
24             }
25             break;
26
27         etc.
28     }
29 }while(rep!=0);
30 printf(" Merci , et au revoir.\n")
31 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), si on cherche à récupérer un élément précis de la liste, dans le pire des cas, il faudra parcourir la liste en entier jusqu’à ce qu’on le trouve.

Dans cette deuxième partie, pour accélérer l’accès à un élément de la bibliothèque, nous allons utiliser une table de hachage. La résolution des collisions se fera par chaînage.

Pour implémenter votre table de hachage, vous pourrez utiliser les struct suivantes (par exemple) :

```

1 typedef struct cell{
2     int clef;
3     /* int num;... Donnees utiles */
4     struct cell *suivant;
5 }cell_t;
6
7 typedef struct{
8     int nE; /*nombre d'elements contenus dans la table de hachage */
9     int m; /*taille de la table de hachage */
10    cell_t **T; /*table de hachage avec resolution des collisions par chainage */
11 } tableHachage_t;

```

Q 2.1 Créez une fonction `tableHachage_t* initTableHachage(int m)`; permettant d’allouer l’espace mémoire nécessaire à votre table de hachage de taille m .

Q 2.2 Fonction clef : La fonction “clef” de la table de hachage à implémenter doit permettre d’associer une valeur numérique au contenu présent dans la table de hachage. Le contenu à stocker est le struct de type `s_livre`, c’est-à-dire un titre, un nom d’auteur et un numéro de livre. Comme fonction clef, vous pouvez utiliser les caractères contenus dans le nom de l’auteur et utiliser la somme des valeurs ASCII de chaque lettre du nom.

Créez la fonction `int fonctionClef(char *nom)`; réalisant cette opération.

Q 2.3 Fonction de hachage : Il est ensuite nécessaire de transformer la clef obtenue en une valeur entière utilisable par la table de hachage (c’est-à-dire entre 0 et m non compris) et permettant d’éviter au maximum les collisions.

Vous pourrez par exemple utiliser la fonction de hachage $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$ pour toute clef k , où $A = \frac{\sqrt{5}-1}{2}$ (nombre d’or diminué de 1, proposé par Donald Knuth ¹). Vous testerez expérimentalement plusieurs valeurs de m afin de déterminer la valeur la plus appropriée.

Il est également possible d’utiliser des fonctions de hachage bien différentes : vous pouvez proposer d’autres fonctions.

Créez la fonction `int fonctionHachage(int clef, int m)`; réalisant cette opération.

Q 2.4 Adaptez les questions 1.2 et 1.3 à l’utilisation d’une table de hachage pour gérer la bibliothèque. Pour insérer un ouvrage dans une bibliothèque, vous devrez procéder en trois étapes :

1. Identifier la clé correspondant à l’ouvrage (grâce à son auteur).

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 .

2. Trouver dans quelle case de la table de hachage insérer l'ouvrage (grâce à la fonction de hachage).
3. Insérer l'ouvrage dans la liste chaînée correspondant à cette case.

Exercice 3 – Comparaison des deux structures

On veut comparer les deux structures (liste et table de hachage) par rapport au temps nécessaire pour effectuer les fonctions de recherche.

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. Quelle structure (liste ou table de hachage) est la plus appropriée pour chacune de ces recherches ?

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 au moins en double 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 000² avec un pas croissant.

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.

N'oubliez pas de rendre votre code ET votre rapport au début de la séance 4, en respectant les consignes du site du module !

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.