Mini-Projet: Gestion d'une bibliothèque

Ce mini-projet s'étale sur deux semaines (séances 2 et 3 de TME). Il est noté et doit être rendu par mail à vos chargés de TD/TME lors du TME numéro 4. A cette occasion, vous lui ferez une démonstration rapide de votre code.

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. Nous utiliserons dans ce projet pour implémenter une bibliothèqe :
- TME2 : une liste simplement chaînées de struct
- TME3 : une table de hachage de struct.

L'ensemble des fichiers nécessaires à ce mini-projet peuvent être récupérés sur le site web de l'UE : `https://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2016/ue/2I006-2017fev/`
Il s'agit :
- d'un fichier GdeBiblio.txt dont le format est très simple : chaque livre (une entrée) correspond à une ligne ; chaque ligne comprend le numéro du livre, son titre et enfin son auteur.
- d'une libraire d'aide à la lecture de fichier entree_sortie.c (et son .h) (son fonctionnement est donné en commentaire).

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 les structures suivantes :

```
1   typedef struct livre{
2     int num;
3     char *titre;
4     char *auteur;
5     struct livre * suiv;
6   } s_livre;
7   typedef struct{
8     s_livre * L;  /*Liste chainee des livres */
9     int nbliv;  /* Nombre de livres dans la liste */
10  } Biblio;
```

Q 1.1 Créer les fichiers correspondants et créer un MakeFile.

Q 1.2 Créer une fonction `intialise_biblio` pour une bibliothèque vide.

Q 1.3 Créer un main en suivant les principes suivant :
- entrée des données en ligne de commande pour le nom du fichier et le nombre de lignes à lire dans

le fichier (voir ci-dessous)
- menu permettant à l'utilisateur d'utiliser les différentes fonctions du programme.
Voici un exemple possible

```
int main(int argc,char* *argv){
  /* argc est le nombre de mot de la ligne de commande
     argv est un tableau de chaines de caracteres:
         une chaine par mot de la ligne de commande  */
  int ch;
  char* nomfic;

  int nlignes;
  Biblio B;
  intialise_biblio(&B);

  if (argc!=3){
    printf("Erreur format: %s <NomFichierBiblio.txt> <NbLigneALire>",argv[0]);
    return 1;
  }

  nomfic=strdup(argv[1]);    /* stdup alloue et copie une chaine de caracteres */
  nlignes=atoi(argv[2]);     /* atoi transforme une chaine de caracteres en entier */

  printf("Lecture:\n");
  lecture_n_entree(nomfic,nlignes,&B);

  do{
    menu();  /* affiche le menu */
    scanf(" %d",&ch);

    switch(ch){
    case 1:
      printf("Affichage\n");
      affichage(&B);
      break;
    case 2:
      /* ETC */
      break;
    }

  }while(ch!=0);

  printf("Au revoir\n");

  return 0;
}
```

Q 1.4 Créez une fonction `void lecture_n_entree(char *nomfic, int n, Biblio *B);` permettant de lire n entrées de ce fichier et de les stocker dans une bibliothèque.
Pour lire facilement les champs du fichier, vous pouvez utiliser les fichiers `entree_sortie.h` et `entree_sortie.c`. Le module `entree_sortie` permet de manipuler facilement des textes en langage C. Vous pouvez lire les commentaires décrivant les fonctions dans le fichier `entree_sortie.h`.

Q 1.5 Créez les fonctions suivantes permettant
- 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
- l'insertion d'un nouvel ouvrage

— la suppression d'un ouvrage
— la recherche des ouvrages au moins en double. Deux ouvrages sont identiques s'ils ont le même auteur et le même titre. Cette fonction devra renvoyer une liste comprenant les ouvrages qui sont au moins en double.

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.

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.

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).
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 50 000 [2] 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.