Bilblio.h :

Définir les structures de Livre et Biblio

Biblio.c :

Définir les fonctions pour allouer un Livre ou un Biblio (creer_livre, creer_biblio),

```
Livre* creer_livre(int num, char* titre, char* auteur);
Biblio* creer_biblio();
```

libérer les espaces occupé par les Livres ou les Biblios(liberer_livre, liberer_biblio)

```
void liberer_livre(Livre* l);
void liberer_biblio(Biblio* b);
```

les opérations pour les élément stocker dans le Biblio(les fonctions suivants)

```
void inserer_en_tete(Biblio* b, int num, char* titre, char* auteur);
void afficher_livre(Livre* l);
void afficher_biblio(Biblio* b);
Livre* chercher_num(Biblio* b, int num);
Livre* chercher_titre(Biblio* b, char* titre);
Biblio* chercher_auteur(Biblio* b, char* auteur);
void supprimer_livre(Biblio* b, int num, char* titre, char* auteur);
Biblio* trouver_exemplaire(Biblio* b);
```

afficher un menu pour les utilisateurs savoir les fonctions dans ce programme

```
void menu();
```

entreeSortieLC.h :

définir deux fonctions pour lire sur un fichier ou sortir les résultat à un ficher

```
Biblio* charger_n_entrees(char* nomfic, int n);
void enregistrer_biblio(Biblio *b, char* nomfic);
```

trouver_exemplaire :

Premièrement, on crée un nouvelle Bilblio pour stocker toutes les données qui sont vérifié avec le standard de la question.

Pour chaque élément dans L (la liste chainée de Biblio qu'on espère de rechercher). Cette méthode est besoin de deux boucles.

Première boucle : accès aux tous les livres stocké dans Biblio, on suppose que cet élément s'appelle A. Et pour la deuxième, il faut accès aux éléments qui ont leurs positions d'après A (connecté par le pointeur « Livre* suiv »). Si on trouve le Livre exemplaire, on le stocke dans la nouvelle bibliothèque qu'on crée précèdent avec l'appel de la fonction « inserer_en_tete » C'est-à-dire que la complexité de cet algorithme est (n -1) + (n -2) + … + 1 + 0 où n est la taille de la bibliothèque, en total, c'est n(n -1) / 2 qui a une complexité temps pire cas en O(n^2)

biblioH.h :

définir les structures de LivreH et BiblioH

biblioH.c :

definir les fonctions pour transformer un nom avec la type char* à la somme des valeur ASCII (fonctionClef)

```
int fonctionClef(char* auteur);
```

Allouer un LivreH ou un BiblioH (creer_livreH, creer_biblioH),

```
LivreH* creer_livreH(int num,char* titre,char* auteur);
BiblioH* creer_biblioH(int m);
```

libérer les espaces occupé par les LivreHs ou les BiblioHs (liberer_livreH, libererBiblioH),

```
void liberer_livreH(LivreH* l);
void liberer_biblioH(BiblioH* b);
```

fonctionHachage :

```
int fonctionHachage(int cle, int m);
```

Cette fonction permet de transformer la cl'e de chaque livre en une valeur entière utilisable par la table de hachage.

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

Où A = $\frac{\sqrt{5}-1}{2}$

Commend.txt :

Le fichier commands.txt peut s'utiliser ainsi :

./main GdeBiblio.txt 20 5 < commands.txt

De même avec le « main » pour les tableaux hachages, en faisant les changements nécessaires.

Nous n'avons pas automatisé la vérification de la correction, donc voici une description des tests contenus dans commands.txt et de l'affichage attendu, dans l'ordre dans lequel ils sont exécutés.

Et après, on utilise gnuplopt -p < commande.txt pour analyser les vitesses de leurs opérations entre les deux structures différents. Cette commande peut procéder les photos pour voir la complexité entre les deux algorithmes.

Les photos sont dans le répertoire qui ont une suffixe .png .

Recherche par titre et par numéro

Comme la clé ne dépend que du nom de l'auteur, nous ne nous attendons pas à une différence substantielle de performances entre la liste chaînée et la table de hachage pour les tests de recherche d'ouvrage par numéro et par titre. En effet, chercher dans la table de hachage équivaut à chercher dans *m* petites listes chaînées, avec un *overhead* supplémentaire dû à la boucle extérieure.

Recherche par auteur

Pour la recherche par auteur, il est raisonnable d'espérer avoir des performances significativement meilleures avec la table de hachage qu'avec la liste chaînée, puisque l'on n'aura plus à scanner toute la bibliothèque, mais seulement les livres dont l'auteur a le même *hash* que celui qu'on cherche. C'est le cas puisque la clé dépend de l'auteur, ce qui permet de savoir dès le début dans quelle case du tableau seront rangés les livres que l'on cherche.

Recherches des ouvrages en plusieurs exemplaires

Voici comment évolue le temps d'exécution de la fonction rechercher_doublons avec une liste chaînée puis avec une table de hachage. Les tests sur la table de hachage ont été fait avec le paramètre $m = \frac{n}{2}$, pour *n* le nombre de livres stockés dans la table de hachage.