TME2 : Langage C: étude de cas sur les tables de hachage

Objectif

Pour développer une application en C, vous devez savoir:

- Ecrire un programme C en respectant des conventions d'écriture.
- Compiler en plusieurs fichiers objet et construire une librairie.
- Ecrire un Makefile.
- Debugger en utilisant gdb ou xgdb.
- Faire des mesures de performances avec gprof.
- Ecrire un man sur l'outil.

L'objectif de ce TME est double :

1. Il doit d'une part vous permettre de complêtez l'auto-évaluation de vos connaissances des outils de developpement C que vous avez commencée dans le précédent TME, en vous posant des questions auxquelles vous devriez savoir répondre. Si ce n'est pas le cas, vous **devez** trouver les réponses dans les documentations (man, web), ou auprès de vos camarades.
2. Il introduit de nouveaux outils permettant l'indentation automatique d'un programme source (outil *indent*), la constructtion d'une bibliothèque C (outil *ar*), ou l'écriture d'une documentation (outil *man*).

Il vous offre également un modèle de programme, avec Makefile et man pour vos futurs développements.

Vous devez commencer par copier sur votre compte le répertoire :

```
$ cp -rp /users/enseig/encadr/cao/tme2 ~/cao/tme2
```

Ce répertoire contient un programme utilisant une table de hachage. Le travail demandé comporte deux phases. Dans un premier temps vous devez analyser le code fourni, et **rédiger** des réponses aux questions portant sur ce code. Dans un deuxième temps, vous devrez modifier ce programme, pour introduire de nouvelles fonctionnalités.

Le programme fourni compte le nombre de mots d'un fichier texte et indique le nombre total de mots dans le fichier et le nombre de mots différents. Vous devrez modifier ce programme de façon à ce qu'il indique, pour chaque mot, les numéros de toutes les lignes où le mot est présent

Vous donnerez également des statistiques sur l'usage des tables de hachage, telles que le nombre moyen de comparaisons nécessaire lors de la recherche d'un mot

Description des sources et principe des tables de hachage

- `Makefile ..... description du processus de construction de l'exécutable.`
- `main.c, main.h ..... programme principal source et déclaration.`
- `count.c, count.h ..... algorithme de parcours d'un fichier texte en vue de comptage.`
- `hte.c, hte.h ..... fonction de création des tables de hachage et déclaration de toutes fonctions de gestion.`
- `dico.c, dejavu.c, namealloc.c .. fonctions de gestion des tables pour trois types d'usage`
- `man1/tool.1 ..... fichier au format man`

Une table de hachage est une structure de données permettant de stocker des ensembles contenant un nombre quelconque d'éléments, où chaque élément est un couple de la forme (clé, valeur). Le plus souvent la clé est une chaîne de caractères. La valeur peut être un nombre ou une structure de données plus ou moins complexe. Le principal objectif d'une table de hachage est d'accélérer la recherche d'un élément par sa clé.

Pour représenter un ensemble de couple (clé, valeur) la méthode la plus simple consiste à les stocker dans une liste chainée. La recherche d'un élément se fait alors par un parcours de la liste, donc de l'ensemble des éléments, ce qui n'est pas très efficace si le nombre d'élément est grand.

Pour accélérer la recherche, on créé un tableau de listes chainées. Le nombre de cases de ce tableau doit être de l'ordre de grandeur du nombre maximal d'éléments que l'on souhaite gérer. Si on veut gérer 1000 éléments, il faut un tableau d'environ 1000 cases. On définit ensuite une fonction, que l'on nomme **fonction de hachage**, qui calcule un index à partir de la clé d'un élément. Un élément doit être rangé dans la liste chainée associée à la case du tableau définie par l'index calculé par la fonction de hachage.

Pour que cette méthode soit efficace, il faut que les éléments se répartissent aussi uniformément que possible dans les différentes cases de la table. Il existe plusieurs méthodes de calcul de l'index, nous vous donnons celle proposé par Donald Knuth. Dans la pratique, il n'est pas possible d'éviter les collisions, et deux éléments ayant des clés différentes peuvent avoir le même index de hachage.

En résumé, pour ranger 1000 éléments, on fabrique un tableau de 1000 listes chainées avec l'espoir que la plupart des listes ne contiendront qu'un seul élément et qu'il n'y aura qu'un tout petit nombre de listes contenant plus d'un élément.

La propriété principale d'une table de hachage est que (si la taille de la table est du même ordre de grandeur que le nombre d'éléments), le temps de recherche est en O(1), c'est à dire indépendant du nombre d'éléments, (même pour un million d'éléments). Les deux principales actions sont l'ajout d'un élément (add) et la recherche d'un élément (get).

- La fonction get() prend en paramètre la clé de l'élément recherché. Si l'élément existe, elle rend la valeur associée.
- La fonction add() prend en paramètre le couple (clé, valeur). Si l'élément existe, elle change sa valeur, sinon elle créé l'élément.

Questions sur le code fourni

Le Makefile

Les premières questions portent sur le fichier attachment:"Makefile"

1. Completez la liste des dépendances pour les cibles : `main.o ... namealloc.o`.
2. Réécrivez les commandes en utilisant les variables automatiques : `$@ $< $^`
   - `$@` : désigne le fichier cible d'une règle.
   - `$<` : désigne le premier fichier de la liste des fichiers source d'une règle.
   - `$^` : désigne la liste des fichiers source d'une règle.
3. Donnez une raison à la definition des commandes et paramètres au début du Makefile
4. A quoi servent les options -p, -g, -wall, -werror, -ansi ?
5. Comment demander l'optimisation maximale du compilateur ?
6. L'option -p est présente dans LDFLAGS et CFLAGS, pourquoi n'est-ce pas le cas de -g ?
7. Que fait la regle indent ? quelle est la signification des flags utilisés par le programme indent ?

Le programme main

Les questions suivantes portent sur le programme principal attachment:main.c

- A quoi sert chaque include ?
- Pourquoi a-t-on un fichier main.h ?
- Expliquez le fonctionnement de la fonction getopt (`man 3 getopt`)

  Ajoutez l'option -h qui affiche l'usage du programme et un petit texte de description du comportement (très court, c'est juste pour l'exercice).
  Vous ajouterez plus tard l'option -s qui demande les statistiques d'usage de la tables de hachage.

- A quoi sert l'appel de return a la fin de la fonction main() ?
- Pourquoi y-a-t-il exit() a la fin de la fonction usage() ?
- Qu'est ce qu'un appel systeme, en voyez-vous dans ce fichier, si oui lesquels,

  citez en d'autres.

- Quelle precaution doit on prendre lors de leur utilisation ?
- Ou sont definiies les fonctions standards ?
- Qu'est-ce qu'un filtre unix ?
- Que faut-il faire pour transformer ce programme en filtre ?

et sur le fichier de prototype associé attachment:main.h

- A quoi servent les lignes les 2 premières lignes et la dernière ?
- Pourquoi inclure `stdio.h` ici ?

Les fonctions de bases de la tables de hachage

Questions sur les déclaration du fichier attachment:hte.h

- Les types `hte_item_t` et `hte_data_t` sont des structures dont le contenu n'est pas défini ici.

  Leur contenu n'est pas défini non plus dans le fichier `hte.c`, en revanche il est défini dans les fichiers `dico.c`, `dejavu.c` et `namealloc.c` Quelle est alors

la contrainte d'usage de ces types dans le fichier `hte.c` ? Quel est l'intérêt de cette écriture ?

- Dans la définition des prototypes de fonctions, le nom des paramètre est-il nécessaire ? si non pour les mettre ?

et sur les fichiers de créations des tables attachment:hte.c

- Faîte un dessin représentatif de la valeur de la variable locale `root` de la fonction `hte_create` si `nb_list==10` juste avant l'appel return.
- La fonction `hte_hash` peut-elle provoquer une erreur de segment ? Comment y remedier proprement ?
- Dans la fonction `hte_create` comment fait-on pour tester le retour de `malloc` ? à quoi celà sert-il ?

Le service *dictionnaire*

Le fichier attachment:dico.c rassemble les fonctions d'accès à une table de hachage utilisé comme dictionnaire.

- Le type `hte_data_t` n'est pas défini ici. Est-ce grave ? Où devra-t-il être défini ?
- Pourquoi teste-on key dans la fonction `hte_get` ?
- Pourquoi définit on la structure `hte_item_s` ici ?
- Dans la structure `hte_item_s` le champ KEY est un tableau de taille indéfinie.

  Quel différence y aurait-il avec une définition du type `char *KEY` ?

- A quoi sert la fonction `perror` ?

Les autres services possibles

Nous vous proposons deux autres services possibles des tables de hachage dans les fichiers, attachment:dejavu.c et attachment:namealloc.c.

Le premier, dejavu, permet de répondre à la question "ai-je déjà vu cette chaine de caractères". Le second, namealloc, permet de garantir que si deux chaines de caractères sont identiques alors elles seront rangées à la même adresse.

Vous n'utiliserez pas ses services au cours de ce TME. Cependant, ils vous seront utiles lors des TME futurs. Vous pourrez faire alors une édition de liens avec la bibliothèque libhte.a. Pour aujourd'hui, contentez-vous de jeter un coup d'oeil par simple curiosité.

L'utilisation du dictionnaire pour compter des occurences de mots

Le code se trouve dans le fichier attachment:count.c

- Le mot clé **static** est utilisé de trois manières différentes. Quel est son effet ?

Evolution du programme

La fonction count fourni permet de faire de détecter les mots à occurences multiples. Nous souhaitons qu'il indique en plus les numéros de lignes où ces occurences apparaissent.

Compte-Rendu