TP noté de programmation impérative

ÉNSIIE, semestre 1

mercredi 4 janvier 2023

Durée : 2h30.
Les exercices 2 et 3 sont indépendants, et peuvent donc être traités dans l'ordre voulu.
Un code qui ne compile pas entraîne une note nulle. La présence de *warnings* à la compilation entraîne une perte de points.

Vous trouverez à l'emplacement `/pub/FISE_PRIM11/TP_note/` des fichiers `hash.h` et `hash.c` contenant une interface et une implémentation des fonctions de hachage. (Cf. explication ci-dessous.) Vous devrez utiliser ces fichiers pour réaliser votre TP, mais vous ne devrez en aucun cas les modifier. Le reste de votre code devra être écrit en accord avec ces fichiers.

Au milieu de la session (15h15), vous déposerez une archive `.tar.gz` contenant l'état courant de votre travail (même s'il ne compile pas encore) dans le dépot `ipi_tp_note_intermediaire_2022` sur http://exam.ensiie.fr/ . Le travail dans ce dépôt ne sera pas noté, mais son absence entraînera une note nulle.

À la fin de la session vous déposerez une archive `.tar.gz` contenant votre travail dans le dépôt `ipi_tp_note_final_2022` sur http://exam.ensiie.fr/ . Seul ce dépôt sera noté. Vous veillerez à ce que votre archive contienne bien les fichiers attendus, aucune erreur ne sera tolérée.

Tous les documents, cours, TD et TP sont autorisés. Les dépôts seront comparés deux à deux. En cas de similitudes entre les codes (y compris, mais pas seulement, si vous vous êtes contenté de recopier un code en changeant le nom des variables et/ou les commentaires), les deux auteurs se verront attribuer la note 0 à l'ensemble du TP, y compris si la similitude ne porte que sur une question. Aucun échange, y compris électronique, n'est autorisé durant l'examen.

Tables de hachage

On rappelle qu'un dictionnaire (aussi appelé tableau associatif ou table d'association, en anglais map) est une structure de données abstraite qui permet de modéliser des fonctions (au sens mathématique) partielles discrètes : on associe à certains éléments d'un ensemble K (les clefs) des valeurs d'un ensemble V.

Nous avons vu en TP qu'une façon simple d'implémenter un dictionnaire est d'utiliser une liste d'associations. L'inconvénient est que la recherche d'un élément se fait dans le pire des cas en temps linéaire par rapport au nombre d'associations.

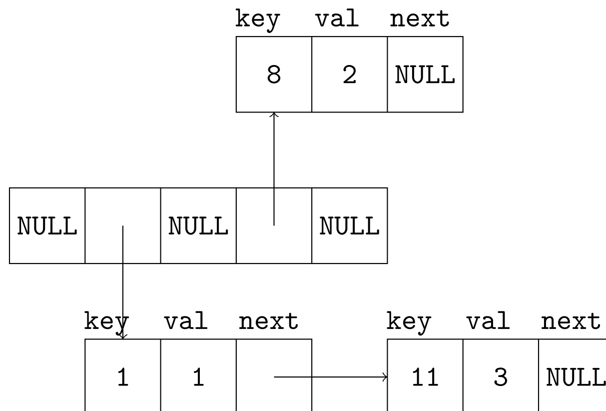
Par contre, l'accès à un élément dans un tableau se fait en temps constant. Le problème ici est que l'ensemble des clefs potentielles (pour ce TP : `int`) est beaucoup trop grand pour envisager d'avoir un tableau avec une case pour chacune d'elle.

Pour pallier ceci, on va utiliser une fonction de hachage : étant donné un entier N fixé, la fonction de hachage h va prendre une clef k et retourner un entier $h(k)$ compris entre 0 et $N-1$. On dit alors que $h(k)$ est la valeur de hachage de k.

On stockera l'association entre la clef k et une valeur v dans la case $h(k)$ d'un tableau de taille N ; ce tableau sera appelé table de hachage.

Comme a priori il y a plus de clefs potentielles que N, il y aura des clefs différentes qui seront associées à la même valeur de hachage. (Autrement dit, on aura $h(k_1) = h(k_2)$ pour $k_1 \neq k_2$.) On parle alors de collision. Une bonne fonction de hachage permet de limiter les collisions, mais ne peut pas les empêcher totalement. Pour pouvoir gérer les collisions, le tableau de la table de hachage ne contiendra pas directement l'association entre clef et valeur, mais il contiendra en fait une liste d'associations. Ainsi, la case d'indice i de la table contiendra la liste des associations dont la valeur de hachage de la clef vaut i.

Par exemple, si $N = 5$, et si la fonction de hachage associe à k le reste de la division euclidienne de k par 5, alors on peut avoir une table de hachage suivante pour stocker les associations $\{1 \mapsto 1; 8 \mapsto 2; 11 \mapsto 3\}$:



Dans le module `hash` vous est fournie une fonction de hachage. Plus précisément,

- la procédure `set_cardinal` permet de fixer le nombre N ; en cas d'utilisation de plusieurs tables de hachage, on s'arrangera pendant ce TP pour utiliser le même N ;
- la fonction `hash` permet de calculer la valeur de hachage d'une clef donnée.

Exercice 1 : Interface

Dans un fichier `hashtbl.h`, écrire l'interface des tables de hachages. On y déclarera donc (sans écrire d'implémentation ici) :

1. des types concrets entiers pour les clefs et les valeurs ;
2. un type abstrait pour les tables de hachage ;
3. une procédure `hashtbl_init` qui prend en paramètre une table de hachage par référence, et un entier N, et qui initialise la table pour qu'elle ait la taille N ;
4. une procédure `hashtbl_add` qui prend en paramètre une clef, une valeur, et une table de hachage par référence, et qui ajoute l'association entre la clef et la valeur dans la table ;
5. une fonction `hashtbl_find` qui prend en paramètre une clef et une table de hachage, et qui retourne la valeur dernièrement associée à la clef dans la table ; si aucune valeur n'est associée à la clef, on retournera `-1` ;
6. une procédure `hashtbl_remove` qui prend en paramètre une clef et une table de hachage par référence, et qui retire de la table la dernière association avec la clef ; sans effet s'il n'existe aucune association avec la clef dans la table ;

7. une procédure `hashtbl_replace` qui prend en paramètre une clef, une valeur, et une table de hachage par référence, et qui remplace la dernière association avec la clef pour celle entre la clef et la valeur.

Pour ce TP, on considérera trois modules :

— le module `hash.c` fourni ;

— un module `hashtbl.c` qui implémente les tables de hachage (exercice 2) ;

— un module `cart.c` qui utilise les tables de hachage pour gérer un panier d'achat (exercice 3).

8. Écrire un fichier `Makefile` pour le projet, que l'on complétera au fur et à mesure de l'avancement. Au final, on veut produire un exécutable appelé `cart_manager` qui réunira ces trois modules. Le module `hash` utilisant la bibliothèque `math.h`, on utilisera l'option `-lm` pour l'édition de lien.

Exercice 2 : Implémentation

Dans un fichier `hashtbl.c`, écrire l'implémentation des tables de hachages.

9. Définir un type pour les listes d'associations.
10. Pour `hashtbl_init`, il faut mettre N à la bonne valeur grâce à `set_cardinal`, et il faut allouer la mémoire nécessaire pour la table ; initialement, chaque case de la table contiendra la liste vide.
11. Pour `hashtbl_add`, on calcule h la valeur de hachage de la clef, puis on ajoute l'association en tête de la liste située dans la case h de la table.
12. Pour `hashtbl_find`, on calcule h la valeur de hachage de la clef, puis on cherche la valeur associée à la clef dans la liste située dans la case h de la table.
13. Pour `hashtbl_remove`, on calcule h la valeur de hachage de la clef, puis on retire de la liste située dans la case h de la table la première occurrence d'une association avec la clef.
14. Pour `hashtbl_replace`, dans un premier temps on pourra se contenter de faire `hashtbl_remove` puis `hashtbl_add` ; dans un deuxième temps on proposera une version qui ne fait pas de nouvelle allocation.

Exercice 3 : Utilisation : panier d'achat

On souhaite utiliser les tables de hachage pour gérer un programme de commerce. On considérera deux tables : la première associera aux références des produits le nombre qu'il en reste en stock ; la deuxième constituera le panier du client et associera aux références des produits leur nombre dans le panier.

Dans un fichier `cart.c` :

15. Écrire une fonction `add_one_to_cart` qui prend en paramètre la table de hachage du stock par référence, la table de hachage du panier par référence, et un entier représentant une référence de produit ; cette fonction doit ajouter un produit de la référence donnée dans le panier, à condition qu'il y en ait encore en stock ; pour cela

on remplacera le nombre éventuellement présent dans le panier par ce nombre plus un ; le stock devra être mis à jour ; on retournera `1` si l'opération a pu se faire, et `0` si le stock n'était pas suffisant.

16. Écrire une procédure `suppress_article` qui prend en paramètre la table de hachage du stock par référence, la table de hachage du panier par référence, et un entier représentant une référence de produit ; cette procédure doit retirer les produits de la référence donnée du panier et les remettre dans le stock ; sans effet s'il n'y a déjà aucun produit de cette référence dans le panier.
17. Écrire une procédure `print_cart_article` qui prend en paramètre la table de hachage du panier et un entier représentant une référence de produit ; cette procédure doit afficher combien le panier contient de produits de la référence donnée ; on affichera de la façon suivante

    ```
    Ref. 0xABCDEF01: 3 item(s)
    ```

    NB : on pourra utiliser le flag `%X` de `printf` pour afficher la référence.
18. Dans la fonction `main`, tester le scénario suivant :
    - on déclare et on initialise deux tables pour le stock et le panier (avec la même valeur pour N) ;
    - on remplit le stock en y ajoutant les associations suivantes :

      0xC0FFEE $\mapsto 4$

      0xC0C0A $\mapsto 2$

      0xDECAF $\mapsto 3$

      0xBA0BAB $\mapsto 1$
    - on essaie d'ajouter successivement deux 0xC0FFEE, un 0xDECAF et deux 0xBA0BAB dans le panier, en affichant un message d'erreur en cas d'échec ;
    - on supprime du panier les 0xC0FFEE ;
    - on affiche le contenu du panier pour 0xC0FFEE, 0xC0C0A, 0xDECAF et 0xBA0BAB.

    Le programme devrait donc afficher :

    ```
    Could not add 0xBA0BAB to cart.
    Ref. 0xC0FFEE: 0 item(s)
    Ref. 0xC0C0A: 0 item(s)
    Ref. 0xDECAF: 1 item(s)
    Ref. 0xBA0BAB: 1 item(s)
    ```