

Devoir surveillé

22 octobre 2019 — Durée 1h15

Document autorisé : **Mémento C** vierge de toute annotation manuscrite

Introduction

Une table clef/valeur est une structure de donnée dans laquelle on peut (i) stocker des couples clefs/valeurs ; (ii) rechercher la valeur associée à une clef ; ou encore (iii) supprimer une clef.

Il s'agit d'une structure de donnée très utilisée car, bien implémentée et dans de bonnes conditions, les trois opérations peuvent se faire en temps quasiment constant.

Dans ce devoir, les clefs et les valeurs seront toujours des entiers (type `int`).

Définition du type `TableCV`

Le paquetage `type_table_clef_valeur` (en Annexe A) définit le type `TableCV` utilisé dans tout ce devoir. Il implémente aussi la fonction `initialiser_table` qui doit être appelée avant la première utilisation d'une table clef/valeur.

Une `TableCV` est un tableau de `MAX_ELEM` cellules. Chaque cellule contient une structure `Paire`. Cette structure contient trois champs : la clef, la valeur associée, ainsi qu'un champ ayant le type `Bit_Validite` qui peut prendre deux valeurs : `PRESENT` ou `ABSENT` et indique si la cellule est actuellement utilisée.

Finalement, le type `ResultatRecherche` est défini. Il s'agit d'une structure utilisée comme valeur de retour pour les opérations de recherche de clef. Cette structure contient un champ `present` du type `Bit_Validite` qui indique si la clef est présente ou absente. Le second champ `valeur` contient la valeur associée à la clef recherchée, le cas échéant (et n'est pas utilisé en cas d'absence).

Éléments de spécification des opérations

Ajout d'un couple $\langle c, v \rangle$: Modifie la table de sorte que la valeur v soit associée à la clef c ;

Suppression d'une clef c : Modifie la table de sorte qu'il n'y ait pas de valeur associée à la clef c ;

Chercher une clef c : Renvoie un `ResultatRecherche` dont le champ `valeur` vaut la valeur v associée à la clef c dans la table (qui reste inchangée). Si la clef est absente, le champ `present` est `ABSENT`.

Le fichier de spécification (fichier d'en-tête `.h`) est fourni en Annexe B.

Organisation du devoir

Le but de ce devoir est :

- De comprendre comment fonctionnent les opérations d'une table clef/valeur. Pour ce faire, on va écrire un programme qui lit une séquence d'opérations à exécuter puis exécute ces opérations. Dans cette partie, on utilisera une implémentation naïve d'une table clef/valeur. (*Première partie*)
- De comprendre quelles sont les propriétés qui doivent être garanties par une implémentation. On cherchera à construire un jeu de tests pour s'assurer que la version naïve y répond bien. (*Seconde partie*)
- On étudiera enfin une deuxième implémentation optimisée. Pour s'assurer que la version optimisée est correcte, on implémentera un oracle qui se basera sur la version naïve. (*Troisième partie*)
- Finalement, on verra comment passer d'une implémentation à l'autre et les modifications que ça implique. (*Dernière partie*)

Première partie

Exercice 1. [4pts]

Fichier de commandes et paquetages associés

On se donne un format de fichier de commandes d'accès à une table clef/valeur. Le format est le suivant :

- La première ligne contient un unique entier n .
- Le reste du fichier contient n lignes de commande d'accès.
- Une ligne de commande d'accès peut prendre trois formes :
 - Un caractère A suivi de deux entiers.
 - Un caractère C suivi d'un entier.
 - Un caractère S suivi d'un entier.

Un exemple d'un fichier de commandes d'accès est fourni en Annexe C. Le paquetage `commande_es` implémente des fonctions pour lire un tel fichier.

Le paquetage `commande_es` utilise les types `Commande` et `Commandes` (notez le pluriel) définis dans le paquetage `commande` :

- Le type `Commande` est une structure qui contient trois champs : (i) un type de commande (qui peut prendre trois valeurs : (a) AJOUTER; (b) SUPPRIMER; et (c) CHERCHER); (ii) une première opérande, qui contient toujours la clef sur laquelle l'opération agit et (iii) une seconde opérande, utilisée seulement pour l'opération AJOUTER qui contient la valeur à associer à la clef fournie en première opérande.
- Le type `Commandes` est une structure qui contient deux champs : (i) le champ `nb_cmds` de type `int` qui contient le nombre de commandes; et (ii) le champ `cmds` qui contient un pointeur de type `Commande*` qui pointe vers un tableau de `nb_cmds` commandes.

Le fichier de spécification du paquetage est fourni en Annexe D.

Question

En utilisant les paquetages `commande_es`, `commande` et `operation_table_clef_valeur_naive`, écrire un programme de test `test_commandes` qui lit une séquence d'accès depuis un fichier, puis exécute cette séquence.

Un exemple d'exécution du programme `test_commandes` utilisant le fichier en Annexe C est donné en Annexe F. Les commandes A et S sont silencieuses (n'affichent rien) et seule la commande C affiche le résultat sur la sortie standard.

Seconde partie

Exercice 2. [3pts]

Quelles propriétés doivent être garanties par implémentation d'une table clef/valeur (quel est le comportement attendu de chaque fonction)?

Indice : on peut utiliser la notion d'*être dans la table* pour un couple et expliquer ce que chaque opération fait en fonction de cette notion d'inclusion.

Exemple : "Propriété 1 : Après l'opération O avec les opérandes o_1, o_2 , les éléments qui sont dans la table sont les éléments e_1, e_i , etc.."

Exercice 3. [5pts]

Les fichiers de commandes d'accès peuvent servir de tests fonctionnels. En partant de votre réponse à la question précédente, décrire comment construire un jeu de tels fichiers pour tester la correction d'une implémentation d'une table clef/valeur.

Remarque : Les tests doivent être des tests fonctionnels.

Troisième partie

Exercice 4. [4pts]

Dans cet exercice, on introduit une version optimisée des opérations sur les tables clefs valeurs. Le but de l'exercice est d'écrire un oracle `oracle` qui vérifie que la version optimisée est correcte en la comparant à la version naïve.

Dans cet exercice, on admet que l'implémentation naïve est correcte.

Le fichier d'en-tête (fichier `.h`) est donné en Annexe G. Les opérations portent le même nom que leur équivalent naïf, avec le suffixe `_opt`.

Remarque : L'implémentation optimisée ne change que l'implémentation des opérations, et pas le type `TableCV`, qui reste commun aux deux implémentations.

Question

Écrire un oracle `oracle` qui vérifie la correction de l'implémentation optimisée en se basant sur l'implémentation naïve.

Indice : on pourra s'inspirer du programme `test_commandes`.

Dernière partie

Exercice 5. [4pts]

Dans cette question, on admettra que le nom des fonctions optimisées est le même que le nom des fonctions naïves. De fait, un programmeur n'a pas besoin de changer le nom de ses appels pour passer d'une implémentation à l'autre.

Décrire les dépendances entre les paquetages pour compiler le programme `test_commandes` de l'exercice 1 (on pourra par exemple dessiner l'arbre de dépendances).

Quelles dépendances changent si l'on veut passer de l'implémentation naïve à l'implémentation optimisée ?

Indiquez les modifications que cela implique sur le Makefile (en Annexe H). Répondez sur votre copie, décrivez uniquement les lignes qui changent dans le Makefile, par exemple : *La ligne 10 devient*

A. Paquetage type_table_clef_valeur

Le fichier type_table_clef_valeur.c n'est pas nécessaire pour ce devoir.

Fichier type_table_clef_valeur.h :

```
1  #ifndef __TYPE_TABLE_CLEF_VALEUR__
2  #define __TYPE_TABLE_CLEF_VALEUR__
3
4  #define MAX_ELEM 20
5
6  typedef enum {PRESENT, ABSENT} Bit_Validite;
7
8  typedef struct {
9      Bit_Validite b;
10     int clef;
11     int valeur;
12 } Paire;
13
14 typedef struct {
15     Paire associations[MAX_ELEM];
16 } TableCV;
17
18 typedef struct {
19     Bit_Validite present;
20     int valeur;
21 } ResultatRecherche;
22
23 void initialiser_table(TableCV* t);
24
25 #endif
```

B. Paquetage operation_table_clef_valeur_naive

Le fichier operation_table_clef_valeur_naive.c n'est pas nécessaire pour ce devoir.

Fichier operation_table_clef_valeur_naive.h :

```
1  #ifndef __OPERATION_TABLE_CLEF_VALEUR_NAIVE__
2  #define __OPERATION_TABLE_CLEF_VALEUR_NAIVE__
3
4  #include "type_table_clef_valeur.h"
5
6  void ajouter(int clef, int valeur, TableCV* t);
7  void supprimer(int clef, TableCV* t);
8  ResultatRecherche chercher(int clef, TableCV t);
9
10 #endif
```

C. Exemple de fichier de commandes

```
1 3
2 A 12 3
3 C 12
4 S 12
```

D. Paquetage commande_es.h

Fichier commande_es.h :

```

1 #include <stdio.h>
2 #include "commande.h"
3
4 // Renvoie un tableau de commandes
5 Commandes lire_commandes(FILE* f);

```

E. Paquetage commande

Fichier commande.h :

```

1 #ifndef _COMMANDE_H
2 #define _COMMANDE_H
3
4 typedef enum {AJOUTER, SUPPRIMER, CHERCHER} Type_Commande;
5
6 typedef struct {
7     Type_Commande tc;
8     int operande1;
9     int operande2;
10 } Commande;
11
12 /* Le champs cmds est un pointeur vers un tableau de 'Commande's.
13  * Le champs nb_cmds est un entier qui indique la taille du tableau. */
14 typedef struct {
15     int nb_cmds;
16     Commande* cmds;
17 } Commandes;
18
19 #endif

```

F. Exemple d'exécution de test_commandes

```

1 moi@ordinateur> ./test_commandes exemple_fichier_acces
2 La clef 12 est associée à la valeur 3

```

G. Paquetage operation_table_clef_valeur_optimisee

Le fichier operation_table_clef_valeur_optimisee.c n'est pas nécessaire pour ce devoir.

Fichier operation_table_clef_valeur_optimisee.h :

```

1 #ifndef __OPERATION_TABLE_CLEF_VALEUR_OPTIMISEE__
2 #define __OPERATION_TABLE_CLEF_VALEUR_OPTIMISEE__
3
4 #include "type_table_clef_valeur.h"
5
6 void ajouter_opt(int clef, int valeur, TableCV* t);
7 void supprimer_opt(int clef, TableCV* t);
8 ResultatRecherche chercher_opt(int clef, TableCV t);
9
10 #endif

```

H. Makefile

Fichier Makefile utilisé dans le devoir :

```

1 C_FLAGS=-g
2 CC=clang
3

```

```
4 all: test_commandes oracle
5
6 commande_es.o: commande_es.c
7
8 test_commandes.o: test_commandes.c operation_table_clef_valeur_naive.h commande.h ↵
    ↵ commande_es.h
9
10 test_commandes: test_commandes.o operation_table_clef_valeur_naive.o commande_es.o ↵
    ↵ type_table_clef_valeur.o
11 $(CC) $^ -o $@
12
13 oracle: oracle.o type_table_clef_valeur.o operation_table_clef_valeur_naive.o ↵
    ↵ operation_table_clef_valeur_optimisee.o commande_es.o
14 $(CC) $^ -o $@
15
16 clean:
17 -rm *.o test_commandes test_es oracle
18
19 %.o:%.c
20 $(CC) $(C_FLAGS) -c $^
```

Remarque : Les flèches rouges (↵ et ↵) ne sont pas présentes dans le Makefile et indiquent que les sauts de ligne ont été rajoutés en raison de la mise-en-page et n'étaient pas présents initialement dans le Makefile.