

Ce TP se déroule sur deux séances (quatre heures). Les étudiants qui se sentent en difficulté devraient réaliser la section 1, uniquement. Les autres peuvent réaliser la section 2 aussi. Des structures de données introduites dans les premiers TP sont réutilisées.

## 1 Petits exercices sur les tables de hachage

### 1.1 Double hachage

**Question 1.** Implanter un module de table de hachage d'entiers, à partir des déclarations suivantes. La technique utilisée est le double hachage. Prendre comme fonction de hachage :

$$h(a) = (h_1(a), h_2(a)) = (a \bmod N, 1 + (a \bmod (N - 1))).$$

Écrire un programme principal, comportant une boucle (boucle *A*), qui lise une suite d'entiers terminée par  $-1$  et qui les enregistre dans la table. À la fin de la boucle, imprimer la table.

```
#define N 11
struct valeur_hachage {
    int h1;
    int h2;
};
typedef struct valeur_hachage fonction_hachage (int);
enum etat_alveole { libre, occupe };
struct alveole {
    enum etat_alveole etat;
    int valeur;
};
struct table_hachage {
    struct alveole tab [N];
    fonction_hachage* h;
    int n;
};
```

**Question 2.** Compléter le module en lui ajoutant une fonction qui recherche si un élément est présent dans la table. Dans le programme principal, ajouter une deuxième boucle (boucle *B*), qui lise une suite d'entiers terminée par  $-1$ , et qui indique, pour chacun d'eux, s'il est ou non présent dans la table.

**Question 3.** Compléter le module en lui ajoutant une fonction qui supprime un élément de la table. Adapter éventuellement les types proposés plus haut. Dans le programme principal, insérer une troisième boucle, entre les boucles *A* et *B*, qui lise une suite d'entiers terminée par  $-1$ , et qui supprime chacun d'eux de la table.

### 1.2 Simple hachage

Dans cette sous-section, réutiliser les listes chaînées introduites au premier TP.

**Question 4.** Adapter le module précédent pour qu'il s'appuie sur une technique de simple hachage. Ranger les entiers ayant même valeur de hachage dans des listes chaînées. Supprimer les champs et les déclarations inutiles. Utiliser la fonction de hachage suivante :

$$h(a) = a \bmod N.$$

**Question 5.** Écrire une fonction qui permette de visualiser la table de hachage de la question précédente, en utilisant le logiciel *dot*. Par exemple, les commandes

```
$ dot -Tpdf hachage.dot -Grankdir=LR -o hachage.pdf
$ evince hachage.pdf
```

produisent l'affichage de la Figure 1 à partir du fichier *hachage.dot* suivant :

```
digraph G {
    tableau [shape=record,label="<f0> 0 | <f1> 1 | <f2> 2 | <f3> 3 | <f4> 4 |
    <f5> 5 | <f6> 6 | <f7> 7 | <f8> 8 | <f9> 9 | <f10> 10"];
    tableau:f1 -> 45;
    45 -> 56;
    tableau:f6 -> 17;
    17 -> 6;
    tableau:f8 -> 8;
    tableau:f9 -> 9;
};
```

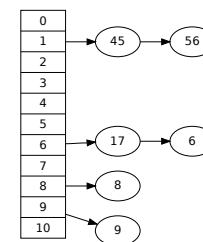


FIGURE 1 – Une table de hachage

**Question 6.** Reprendre la section 1, en réalisant une table de hachage de chaînes de caractères. Réutiliser le type *chaîne* introduit au TP 1.

Toute la difficulté tient au fait que le type *chaîne* est muni d'un constructeur et d'un destructeur (ce qui n'est pas le cas des types *int*, *double* et *char*).

Vérifier avec *valgrind* que vos programmes ne perdent pas de mémoire.

## 2 Algorithmme de l'éditeur de liens

### 2.1 Introduction

Considérons les deux fichiers C suivants.

```
/* Fichier add.c */
static int add_local (int x, int y)
{
    return x + y;
}
int add (int x, int y)
{
    return add_local (x, y);
}

/* Fichier main.c */
extern int add (int, int);
int main ()
{
    int z;
    z = add (3, 4);
    return 0;
}
```

Compilons-les et analysons-les avec la commande `nm`.

```
$ gcc -c add.c main.c
$ nm add.o
000000000000000015 T add
000000000000000000 t add_local
$ nm main.o
                 U add
000000000000000000 T main
```

La lettre qui précède les symboles indique leur type : une lettre minuscule indique un symbole local ; une lettre majuscule différente de `U` indique un symbole global ; la lettre `U` indique une référence indéfinie. Le fichier objet `add.o` contient un symbole global, `add`, et un symbole local `add_local` (local car précédé de `static`). Le fichier objet `main.o` contient un symbole global, `main`, et une référence indéfinie à `add`. L'une des tâches de l'éditeur de liens consiste à résoudre les références indéfinies, en les faisant correspondre à un symbole global de même nom.

L'édition des liens entre un ensemble de fichiers objets est réussie si toutes les références indéfinies sont résolues, si le symbole `main` est défini et s'il n'existe pas deux symboles globaux portant le même nom.

### 2.2 Travail à faire

On souhaite écrire un programme qui reproduise la gestion de la table des symboles d'un éditeur de liens. Le programme doit prendre sur sa ligne de commande une liste de fichiers objets et indiquer si l'édition des liens est réussie ou pas.

Proposer une structure de données (fondée sur l'utilisation de tables de hachages) pour gérer la table des symboles.

Écrire le programme.

### 2.3 Récupération des symboles d'un fichier objet

Pour obtenir les symboles d'un fichier objet `fichier.o`, le plus simple consiste à appliquer la commande `nm` sur le fichier (fonction `system` de la bibliothèque standard du C), à en rediriger la sortie sur un filtre approprié (cf. cours de *système*) pour ne garder que le type et le nom, et à

rediriger la sortie du filtre vers un fichier `fichier.sym`). Le fichier `fichier.sym` peut ensuite être lu en exécutant des instructions inspirées du code C suivant :

```
#include <stdio.h>

int main ()
{
    FILE* f;
    int n;
    char type, ident [32];

    f = fopen ("main.sym", "r");
    n = fscanf (f, "%c %s\n", &type, ident);
    while (n == 2)
    {
        printf ("type = %c, ident = %s\n", type, ident);
        n = fscanf (f, "%c %s\n", &type, ident);
    }
    fclose (f);
}
```

### 2.4 Les références aux fonctions de la bibliothèque standard

Tout programme réaliste contient des références indéfinies vers des symboles de la bibliothèque standard (fonctions `printf`, `malloc`, etc.). Une façon simple de résoudre cette difficulté consiste à se fabriquer un faux fichier objet contenant les symboles des fonctions utilisées et à ajouter ce fichier à la ligne de commande.

Une solution plus élégante consiste à aller chercher les définitions de symboles dans la bibliothèque standard (habituellement `/usr/lib/libc.a`). Attention : bien qu'une bibliothèque ne soit rien d'autre qu'une collection de fichiers objets, le mécanisme d'inclusion des symboles est un peu différent de celui présenté ci-dessus (cf. support de cours).