**Polytech'Lille — GIS 3 — Structures de Données — Feuille de TP numéro 6**

---

Ce TP se déroule sur deux séances (quatre heures). La section 1 consiste uniquement à comprendre les mécanismes de base du simple hachage. La section 2 montre comment adapter ces mécanismes dans le cadre d'une application (la même que dans le TP sur les ABR). La section 3 (réservée aux étudiants les plus rapides) consiste à tout refaire avec la méthode du double hachage.

# 1 Implantation d'une table de simple hachage

On veut réaliser un module dédié aux tables de hachage simples, qui permette de faire fonctionner le programme principal suivant. Les valeurs sont des doubles.

```
/* main.c */

#include <stdio.h>
#include "hachage_simple.h"

int hachage_basique (double d)
{
    return (int)d % N;
}

int main ()
{   struct table T;
    double x;

    init_table (&T, &hachage_basique);
    scanf ("%lf", &x);
    while (x != -1)
    {   enregistrer_table (&T, x);
        imprimer_table (&T);
        scanf ("%lf", &x);
    }

    clear_table (&T);
    return 0;
}
```

Le fichier d'entête `hachage_simple.h` est imposé (Figure 1).

**Question 1.** Réaliser le logiciel. Réutiliser le module `liste_double` écrit lors d'un TP précédent (supprimer les fonctions inutilisées et rejouter les fonctions nécessaires). Consigne : le code écrit dans `hachage_simple.c` doit être totalement indépendant de l'implantation des listes.

**Question 2.** Modifier le programme principal afin de tester la recherche d'un élément.

**Question 3.** Modifier la fonction de hachage de façon à ce qu'elle retourne toujours la valeur 1. Faut-il recompiler le module `hachage_simple` ? Le programme fonctionne-t-il toujours ? Expliquer.

```
/* hachage_simple.h */

#include "liste_double.h"
#include <stdbool.h>

struct alveole {
    struct liste_double L;
};

typedef int fonction_hachage (double);

#define N 10

struct table {
    struct alveole tab [N];
    fonction_hachage* hash;
};

extern void init_table (struct table*, fonction_hachage*);
extern void clear_table (struct table*);
extern void enregistrer_table (struct table*, double);
extern bool rechercher_table (struct table*, double);
extern void imprimer_table (struct table*);
```

FIGURE 1 – Fichier d'entête imposé pour le hachage simple de doubles.

## 2 Dictionnaire Esperanto—Français

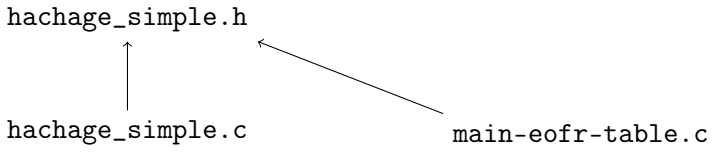


FIGURE 2 – Représentation graphique du traducteur à réaliser.

Dans cette section, on montre une utilisation des tables de hachage dans le cadre d'une application (Figure 2), où les valeurs sont des couples (*clef*, *donnée satellite*). On en profite aussi pour étudier le comportement des tables de hachage sur des données un peu volumineuses : un dictionnaire Esperanto-Français, obtenu à partir du site web `http://purl.org/net/panorama/vortaro/eofr.htm`, avec l'aimable autorisation de M. Godivier. La commande suivante permet de télécharger ce dictionnaire :

```
$ wget http://cristal.univ-lille.fr/~boulier/polycopies/SD/Esperanto-Francais.utf8
```

## 2.1 Modification du module de listes

Les clefs et les données satellites sont des chaînes de caractères, comportant des lettres accentuées (la clef est en Esperanto, la donnée satellite est en Français). On utilise pour cela les *wide char* du langage C, décrits dans la section 2.1 de la feuille de TP 1.

Les étudiants qui le souhaitent peuvent utiliser le type `struct chaine` du TP 1 (il faut alors réinterpréter et adapter le reste de la section). Les autres doivent utiliser le type `wstring` (cf. TP sur les ABR). L'adaptation du module `liste_double` en un module `liste_wstring` demande un peu de soin. On impose le fichier d'entête suivant donné Figure 3.

```
/* liste_wstring.h */

#include <wchar.h>
#define MAXLEN 80
typedef wchar_t wstring [MAXLEN];

struct maillon_wstring
{   wstring clef;
    wstring satellite;
    struct maillon_wstring* next;
};

struct liste_wstring
{   struct maillon_wstring* tete;
    int nbelem;
};

extern void init_liste_wstring (struct liste_wstring*);
extern void clear_liste_wstring (struct liste_wstring*);
extern void ajouter_en_tete_liste_wstring (struct liste_wstring*, wstring, wstring);
extern wchar_t* rechercher_liste_wstring (struct liste_wstring*, wstring);
extern void imprimer_liste_wstring (struct liste_wstring*);
```

FIGURE 3 – Fichier d'entête d'un module de listes adaptées au problème du traducteur.

Chaque maillon comporte trois champs : la clef, la donnée satellite et un pointeur vers le maillon suivant. La fonction de recherche est paramétrée par une liste et une clef. Elle retourne l'adresse de la donnée satellite si la recherche aboutit, et `(wchar_t*)0` sinon.

**Question 4.** Adapter `liste_double.c` en `liste_wstring.c`. Pour l'impression, attention à la remarque ci-dessous, déjà énoncée au TP précédent. Utiliser les fonctions `wcscmp` et `wcscpy` pour les comparaisons et les copies de `wstring`. Pour les affichages, utiliser `wprintf`.

## 2.2 Modification du module de table de hachage

**Question 5.** Adapter le module `hachage_simple` pour l'adapter aux besoins du traducteur. Les modifications sont assez naturelles. Attention aux affichages (remplacer tous les appels à `printf` par des appels à `wprintf`). Deux fonctions voient leur prototype évoluer plus que les autres :

```
void enregistrer_table (struct table* T, wstring clef, wstring satellite);
wchar_t* rechercher_table (struct table* T, wstring clef);
```

### 2.3 Modification du programme principal

Le programme principal donné Figure 4 permet de charger le dictionnaire dans la table de hachage `T`. Il peut être téléchargé par la commande suivante :

```
$ wget http://cristal.univ-lille.fr/~boulier/polycopies/SD/main-eofr-table.c
```

**Question 6.** (optionnelle) Ajouter à ce programme des instructions permettant de rechercher des mots Esperanto et de donner leur traduction.

### 2.4 Expérimentations

On cherche une fonction de hachage qui donne de bons résultats sur ce dictionnaire. Idéalement, toutes les listes de la table devraient contenir le même nombre d'éléments à 1 près.

**Question 7.** Mettre au point une fonction qui affiche de façon synthétique la répartition des entrées dans les listes (l'affichage doit rester lisible pour $N \simeq 1000$).

**Question 8.** Pour différentes valeurs de $N$ (entre 10 et 1000), étudier cette répartition.

**Question 9.** Tester d'autres fonctions de hachage et déterminer celle qui semble la meilleure.

## 3 Double hachage

Reprendre les deux sections précédentes avec des tables de hachage gérées avec la technique du double hachage. Prendre comme fonction de hachage :

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

Des déclarations possibles sont les suivantes :

```
#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;
};
```

```
/* main-eofr-table.c */

#include <locale.h>
#include <wctype.h>
#include <assert.h>
#include <stdio.h>
#include "hachage_simple.h"

int hachage_basique (wstring clef)
{   wint_t result;
    int i;
    result = 0;
    for (i = 0; clef [i] != L'\0'; i++)
        result = (result + (wint_t)clef [i]) % (wint_t)N;
    return (int)result;
}

int main ()
{   struct table T;
    wstring clef;
    wstring satellite;
    wint_t c;
    FILE* f;

    assert (setlocale (LC_ALL, "C.UTF-8") != NULL);
//  f = fopen ("Esperanto-Francais.utf8", "r");
    f = fopen ("Esperanto-Francais.demo", "r");
    assert (f != (FILE*)0);

    init_table (&T, &hachage_basique);
    c = fgetwc (f);
    while (c != WEOF)
    {   int i = 0;
        while (c != L':')
        {   clef [i++] = c;
            c = fgetwc (f);
        }
        clef [i] = L'\0';
        c = fgetwc (f);
        i = 0;
        while (c != L'\n')
        {   satellite [i++] = c;
            c = fgetwc (f);
        }
        satellite [i] = L'\0';
        enregistrer_table (&T, clef, satellite);
        c = fgetwc (f);
    }
    fclose (f);
    imprimer_table (&T);
    clear_table (&T);
    return 0;
}
```

FIGURE 4 – Programme principal chargeant le dictionnaire `Esperanto-Francais.utf8` (ou plutôt un extrait d'une dizaine de lignes) dans une table initialement vide.