

# Programmation Procédurale

## Feuille 7

### Tables de hachage

## 1 Introduction

Pour cette feuille nous voulons implémenter le type abstrait de données `HashTable`. Les tables que nous allons représenter auront

- une clé qui est une chaîne de caractères
- une valeur associée qui peut être de n'importe quel type (et qui sera donc représentée par un *void \**).

### 1.1 Le fichier `hash.h`

Le fichier `hash.h` définit les types et les fonctions exporté par le module `hash.c`. Ce fichier est donné ci-dessous:

```
//                                                    /*- coding: utf-8 *-
//
// hash.h      -- Implementation des tables de hachage
//
#ifndef _HASH_H_
# define _HASH_H_

typedef struct hash_table  *HashTable;    // La table de hachage

// Allocation d'une nouvelle table de hachage vide.
// La valeur donnée en paramètre indique la taille de la table à sa
// création.
HashTable hash_new(int size);

// Chercher la valeur associée à la clé "key" dans "ht". Si "key" est
// présent dans la table, cette fonction renvoie la valeur associée à
// cette clé. Dans le cas contraire, cette fonction renvoie la valeur
// NULL
void *hash_find(const HashTable ht, const char *key);

// Chercher la valeur associée à la clé "key" dans "ht". Si "key" est
// présent dans la table, cette fonction renvoie un pointeur sur la
// valeur associée à cette clé. Dans le cas contraire, cette fonction
// renvoie la valeur NULL
void **hash_find_reference(const HashTable ht, const char *key);

// Ajout de la valeur "value" associée à la clé "key" dans "ht". Si
// "key" est déjà présent on affiche un message d'erreur et on
// sort. Sinon, une nouvelle entrée est créée pour "key".
```

```

void hash_add(HashTable ht, const char *key, const void *value);

// Impression de la table "ht" sur le fichier standard de sortie. Pour
// chaque élément, on affiche que sa clé et l'indice où il est rangé
// dans la table. Un exemple d'exécution est donné ci-dessous:
//
//      Hash table de taille 10
//      0:  'dix'
//      1:  'neuf' 'cinq'
//      2:  'trois'
//      3:  'deux'
//      5:  'six'
//      7:  'huit' 'sept' 'un'
//      8:  'quatre'
//
// Ici, on voit que la case d'indice 4 n'est pas utilisée alors que
// celle d'indice 1 comporte deux clés: "neuf" et "cinq".
void hash_print(const HashTable ht);

// Appliquer la fonction utilisateur "func" à tous les éléments de la table "ht"
void hash_apply(const HashTable ht, void (*func)(const char *key, const void *value));

// Libérer la hash table. Attention le contenu de la table lui même
// n'est pas libéré. C'est à l'utilisateur de le faire
void hash_free(HashTable *ht);

#endif // _HASH_H_

```

## 2 Une première implementation

Etapes:

1. Définir la structure de données `struct hash_table`.
2. Écrire la fonction de création de table `hash_new`.
3. Écrire la fonction de hachage `hash_value`. (si `key` est la suite de caractères  $[c_0, c_1, c_2, \dots, c_n]$ , la valeur renvoyée par cette fonction est  $H = c_0 + 2 * c_1 + 3 * c_2 + \dots, + (n + 1) * c_n$ .  
  
**Note:** cette fonction n'est pas une bonne fonction de hachage, mais on s'en contentera ici.  
Étant donnée cette fonction, comment calcule t'on l'indice de la case où se trouve la liste des mots ayant cette valeur de hachage dans une table de taille `n`?
4. Le fichier `hash.h` propose deux fonctions de recherche. La fonction `hash_find_reference` renvoie un pointeur sur la valeur associée à une clé donnée (et non pas directement la valeur). Pourquoi une telle fonction est elle nécessaire?  
Écrire les deux fonctions de recherche `hash_find_reference` et `hash_find`.
5. Écrire la fonction `hash_add`.
6. Écrire la fonction `hash_print`.
7. Écrire la fonction `hash_apply`.
8. Le module `hash.c` propose une fonction pour détruire une table `hash_destroy`. Comment l'utilisateur doit il l'appeler si il veut que la clés de la table **mais** aussi les valeurs qui leur sont associées soit détruites.

### 3 Seconde implementation

Modifier votre implémentation pour une utiliser une table dont la taille varie: la table sera agrandie lorsque son taux de remplissage sera égal à 75%. Sa taille sera alors multipliée par 1.5.