



ISIMA 1^{ère} ANNEE

Rapport

TP4 de Structure de Données

Anaïs DARRICARRERE
Nada BOUTADGHART

13 mars 2020

Table des matières

1	Présentation générale	1
1.1	Description de l'objet du TP	1
1.2	Description des structures	1
1.2.1	Structure d'une table de hachage indirect	1
1.3	Organisation du code source	2
1.3.1	Les fichiers d'entête	2
1.3.2	Les modules	2
2	Détails des codes sources de chaque fichier	3
2.1	Lch.h	3
2.2	Lch.c	4
2.3	Table.h	7
2.4	Table.c	8
2.5	Main.c	16
3	Compte rendu d'exécution	17
3.1	Jeux de test	17
3.2	Makefile	20

Partie 1

Présentation générale

1.1 Description de l'objet du TP

Le but de ce TP est de réaliser un gestionnaire de dictionnaire bilingue. Ce dernier contient l'ensemble des mots d'un fichier donné en entrée et leurs traductions.

1.2 Description des structures

1.2.1 Structure d'une table de hachage indirect

Une table est une structure construite sur la base d'une table majeure de taille 29 et d'une fonction de hachage.

La table majeure est une liste contiguë de cellules contenant le compteur du nombre d'éléments présents dans la sous-table correspondante et le pointeur de tête de cette sous-table. Une sous-table est une liste chaînée constituée de blocs de 3 mots :

- le 1^{er} contient le mot dans la langue d'origine ;
- le 2nd contient la traduction du mot ;
- le dernier contient le pointeur sur la cellule suivante.

Cette structure est utilisée pour construire un dictionnaire bilingue à partir d'un fichier d'entrée contenant la liste de chaque mot séparé de sa traduction par un ";".

La fonction de hachage utilisée est celle de D.J. Bernstein (cf. `hash_string` code source `Table.c`).

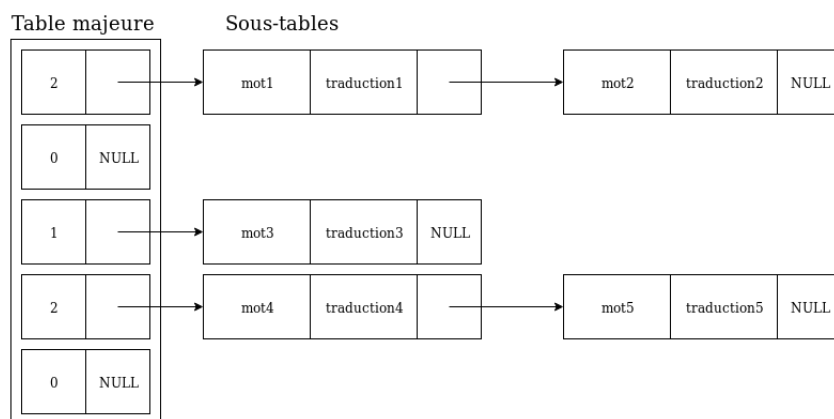


Schéma de structure d'une table de hachage indirect

1.3 Organisation du code source

1.3.1 Les fichiers d'entête

Lch.h contient :

- Des directives de préprocesseur permettant d'inclure les bibliothèques `<stdio.h>`, `<stdlib.h>` et `<string.h>` ;
- La déclaration du type `cellule_t` (pour une cellule de liste chaînée) ;
- Les prototypes des fonctions de gestion de liste chaînée ;

Table.h contient :

- Des directives de préprocesseur permettant d'inclure les bibliothèques `<stdio.h>`, `<stdlib.h>`, `<string.h>`, `<ctype.h>` et `"Lch.h"` le fichier d'entête de la gestion de la liste chaînée écrit au TP1 ;
- La déclaration de la structure `cellule_table` (pour une cellule de la table majeure) ;
- Les prototypes des fonctions de gestion d'une table ;

1.3.2 Les modules

Lch.c contient :

- Une directive de préprocesseur permettant d'inclure le fichier d'entête `"Lch.h"` ;
- Les codes des fonctions de gestion de listes chaînées déclarés dans le fichier `Lch.h` ;

Table.c contient :

- Une directive de préprocesseur permettant d'inclure le fichier d'entête `"Table.h"` ;
- Les codes des fonctions de gestion d'un arbre déclarés dans le fichier `Table.h`, à savoir :
 - `hash_string(str)` : la fonction de hachage appliquée sur une chaîne de caractère ;
 - `INIT_TABLE(maj)` : initialise une table majeure vide ;
 - `RECHERCHE(maj, word)` : recherche la traduction du mot donné en paramètre dans le dictionnaire ;
 - `INSERT_CELL(nouv, maj)` : insert un bloc d'adresse `nouv` dans une sous-table du dictionnaire ;
 - `DICTIONNAIRE(nom_fichier, maj)` : crée une table à partir d'un fichier d'entrée, cette table correspond au dictionnaire ;
 - `TRADUCTION(maj, exp)` : retourne la traduction d'un ensemble de mots ou d'expressions (si un mot est absent du dictionnaire, le mot retourné est celui dans la langue d'origine) ;
 - `COMPT_MOY(maj)` : retourne le nombre moyen de cellules contenues dans les sous-tables du dictionnaire d'entrée ;
 - `SUPP_TABLE(maj)` : supprime et libère la table ;

Main.c contient :

- Une directive de préprocesseur permettant d'inclure les fichiers d'entête `"Table.h"` ;
- Le code du programme principal qui teste les fonctions de gestion de table ;

Partie 2

Détails des codes sources de chaque fichier

2.1 Lch.h

```
/*-----*/
/*                      TP4 - Gestion de liste chaînée                      */
/*                      Fichier d'entête Lch.h                              */
/*-----*/
/* Déclaration des structures et prototypes des fonctions de gestion de listes chaînées */
/*-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*-----*/
/* Déclaration de la structure cellule_t : bloc de la liste chaînée contenant : */
/* mot          Chaîne de caractère désignant le mot à traduire                */
/* trad         Chaîne de caractère désignant le mot traduit                  */
/* suivant      Pointeur vers la cellule suivante                            */
/*-----*/

typedef struct cellule
{
    char * mot, * trad;
    struct cellule * suivant;
}cellule_t;

/*-----*/
/* Prototypes des fonctions */
/*-----*/

cellule_t * CREER_CELL(char * mot, char * trad);

void ADJ_CELL(cellule_t ** prec, cellule_t * nouv);

void SUPP_CELL(cellule_t ** prec);

void SUPP_LCH(cellule_t ** t);
```

2.2 Lch.c

```
/*-----*/
/*                      TP4 - Gestion de liste chaînée                      */
/*                      Lch.c                                              */
/*                                                                    */
/*                                                                    */
/* Fichier contenant le code des fonctions de gestions de listes chaînées */
/*-----*/

#include "Lch.h"

/*-----*/
/* CREER_CELL           Création d'une cellule de type cellule_t           */
/*                                                                    */
/* Algorithme de principe:                                              */
/* - Allouer un bloc de type cellule_t                                  */
/* - Si l'allocation est réussie :                                      */
/*   - Copier les valeurs entrées en paramètre dans la                 */
/*     nouvelle cellule                                                 */
/* FIN                                                                  */
/*                                                                    */
/* Lexique                                                            */
/*                                                                    */
/* En entrée : mot           Chaîne de caractère désignant le mot à traduire */
/* trad           Chaîne de caractère désignant le mot traduit           */
/*                                                                    */
/* Variables intermédiaires :                                          */
/* lg_mot         Longueur du mot à traduire                            */
/* lg_trad        Longueur du mot traduit                               */
/*                                                                    */
/* En sortie : nouv         Pointeur sur la cellule créée               */
/*-----*/
```

```

cellule_t * CREER_CELL(char * mot, char * trad)
{
    cellule_t * nouv = NULL;
    int lg_mot, lg_trad;
    nouv = (cellule_t *)malloc(sizeof(cellule_t));
    if (nouv != NULL)
    {
        lg_mot = strlen(mot);
        nouv->mot = (char *)malloc((lg_mot+1)*sizeof(char));
        if(nouv->mot != NULL)
        {
            lg_trad = strlen(trad);
            nouv->trad = (char *)malloc((lg_trad+1)*sizeof(char));
            if(nouv->trad != NULL)
            {
                strcpy(nouv->mot, mot);
                strcpy(nouv->trad, trad);
                nouv->suivant = NULL;
            }
            else
            {
                free(nouv->mot);
                nouv->mot = NULL;
                free(nouv);
                nouv = NULL;
            }
        }
        else
        {
            free(nouv);
            nouv = NULL;
        }
    }
    return nouv;
}

```

```

/*-----*/
/* ADJ_CELL          Adjonction d'une cellule dans une liste chaînée          */
/*                                                           */
/* Algorithme de principe:                                     */
/*   - Si la cellule à insérer existe :                       */
/*     - Faire pointer la nouvelle cellule sur la cellule suivante */
/*     - Faire pointer la cellule précédente sur la nouvelle cellule */
/*   FIN                                                       */
/* Lexique                                                     */
/* En entrée : prec      Pointeur sur le précédent           */
/*              nouv      Pointeur sur la cellule à insérer   */
/*-----*/

```

```

void ADJ_CELL(cellule_t ** prec, cellule_t * nouv)
{
    if (nouv != NULL)
    {
        nouv->suivant = *prec;
        *prec = nouv;
    }
}

```



```

/*-----*/
/* SUPP_CELL                Supprime une cellule dans une liste chaînée */
/*                          */
/* Algorithme de principe:  */
/*                          - Faire pointer la cellule précédente vers la cellule suivante */
/*                          - Libérer la cellule courante */
/*                          FIN */
/*                          */
/* Lexique                  */
/*                          */
/* En entrée : prec        Pointeur sur le précédent de la cellule à supprimer */
/*                          */
/* Variables intermédiaires : */
/* cour                  Pointeur sur la cellule à supprimer */
/*-----*/

void SUPP_CELL(cellule_t ** prec)
{
    cellule_t * cour = *prec;
    *prec = cour->suivant;
    free(cour->mot);
    free(cour->trad);
    free(cour);
}

/*-----*/
/* SUPP_LCH                Libère une liste chaînée */
/*                          */
/* Algorithme de principe:  */
/*                          - Tant que le pointeur de tête n'est pas NULL : */
/*                          - Libérer la première cellule */
/*                          FIN */
/*                          */
/* Lexique                  */
/*                          */
/* En entrée : t            Pointeur de tête de la liste chaînée */
/*-----*/

void SUPP_LCH(cellule_t ** t)
{
    cellule_t ** prec = t;
    while(*prec != NULL)
    {
        SUPP_CELL(prec);
    }
}

```

2.3 Table.h

```
/*-----*/
/*          TP4 - Gestion de table          */
/*          Fichier d'entête Table.h        */
/*                                          */
/* Déclaration des structures et des fonctions de gestion d'une table */
/*-----*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "Lch.h"

/*-----*/
/* Définition de constante de la fonction de hachage représentant la taille de la table majeure */
/*-----*/

#define HASH_MAX 29

/*-----*/
/* Déclaration de la structure cellule_table : bloc de la liste chaînée contenant : */
/*      compteur      Compteur du nombre de cellules dans la sous-table */
/*      pointeur      Pointeur de tête de la sous-table */
/*-----*/

typedef struct cellule_table
{
    int compteur;
    struct cellule * pointeur;
}cellule_T;

/*-----*/
/* Prototypes des fonctions */
/*-----*/

unsigned int hash_string(const char * str);

void INIT_TABLE(cellule_T ** maj);

char * RECHERCHE(cellule_T * maj, char * word);

void INSERT_CELL(cellule_t * nouv, cellule_T * maj);

int DICTIONNAIRE (char * nom_fichier, cellule_T ** maj);

char * TRADUCTION(cellule_T * maj, char * exp);

float COMPT_MOY(cellule_T * maj);

void SUPP_TABLE(cellule_T * maj);
```

2.4 Table.c

```
/*-----*/
/*                                TP4 - Gestion de table                                */
/*                                Table.c                                              */
/*-----*/

#include "Table.h"

/*-----*/
/* hash_string                    Fonction de hachage appliquée sur une chaîne de caractères */
/*                                (fonction de hachage de D.J. Bernstein)                */
/*-----*/

unsigned int hash_string(const char * str)
{
    unsigned int hash = 5381;
    const char * s;
    for (s = str; *s; s++)
        hash = ((hash << 5) + hash) + tolower(*s);
    return (hash & 0x7FFFFFFF)%HASH_MAX;
}

/*-----*/
/* INIT_TABLE                    Initialisation de la table majeure vide                */
/*                                */
/* Algorithme de principe:                */
/*                                - Pour chaque cellule de la table faire :                */
/*                                - Mettre à 0 le compteur de la cellule                */
/*                                - Initialiser à NULL chacun des mots de la table        */
/*                                FIN                */
/*                                */
/* Lexique                */
/*                                */
/* En entrée : maj                Adresse de la table majeure                */
/*-----*/

void INIT_TABLE(cellule_T ** maj)
{
    int i;
    *maj = (cellule_T *)malloc(HASH_MAX*(sizeof(cellule_T)));
    if (maj != NULL)
    {
        for (i=0; i<HASH_MAX; i++)
        {
            (*maj + i)->compteur = 0;
            (*maj + i)->pointeur = NULL;
        }
    }
}
```

```

/*-----*/
/* RECHERCHE           Recherche de la traduction d'un mot dans le dictionnaire d'entrée */
/*                                                             */
/* Algorithme de principe:                                     */
/* - On initialise le pointeur courant sur la première cellule de la */
/*   sous-table contenant le mot donné en entrée                 */
/*   (utilisation de la fonction de hachage)                     */
/* - On initialise une variable trad à la chaîne de caractère vide */
/* - Tant qu'on est pas à la fin de la sous-table et que la cellule */
/*   courante ne contient pas le mot faire :                     */
/*   - On avance le pointeur du courant au bloc suivant dans la */
/*     liste chaînée                                             */
/* - Si on a trouvé le mot alors                                 */
/*   - On sauvegarde le mot traduit correspondant               */
/* FIN                                                           */
/*                                                             */
/* Lexique                                                       */
/* En entrée : maj       Adresse de la table majeure           */
/*              word      Chaîne de caractère contenant le mot recherché */
/* Variables intermédiaires :                                     */
/* cour          Pointeur sur la cellule courante               */
/* En sortie : trad      Chaîne de caractère vide si le mot n'est pas trouvé et contenant le */
/*                      mot traduit sinon                       */
/*-----*/

```

```

char * RECHERCHE(cellule_T * maj, char * word)
{
    cellule_t * cour;
    char * trad = NULL;
    cour = (maj + hash_string(word))->pointeur;
    while ((cour != NULL) && (strcmp(cour->mot, word) != 0))
    {
        cour = cour->suivant;
    }
    if (cour != NULL)
    {
        trad = (char *)malloc((strlen(cour->trad)+1)*sizeof(char));
        strcpy(trad, cour->trad);
    }
    return trad;
}

```

```

/*-----*/
/* INSERT_CELL          Insertion d'une cellule dans une sous-table */
/* */
/* Algorithme de principe: */
/*      - Calculer l'indice de début de la sous-table */
/*      (utilisation de la fonction de hachage) */
/*      - Insérer la nouvelle cellule en tête de cette sous-table */
/*      (sous-programme ADJ_CEL) */
/*      - Incrémenter le compteur de la sous-table */
/*      FIN */
/* */
/* Lexique */
/* */
/* En entrée : nouv      Pointeur sur la cellule à insérer */
/*      maj      Adresse de la table majeure */
/* */
/* Variables intermédiaires : */
/*      prec      Pointeur sur le précédent */
/*-----*/

void INSERT_CELL(cellule_t * nouv, cellule_T * maj)
{
    cellule_t ** prec;
    unsigned int indice = hash_string(nouv->mot);
    prec = &((maj + indice)->pointeur);
    ADJ_CEL(prec, nouv);
    ((maj + indice)->compteur) = ((maj + indice)->compteur) + 1;
}

/*-----*/
/* DICTIONNAIRE          Création d'un dictionnaire à partir d'un fichier d'entrée */
/* */
/* Algorithme de principe: */
/*      - Ouvrir le fichier en lecture */
/*      - Si le fichier est ouvert : */
/*      - Initialiser la table majeure */
/*      - Tant que la fin du fichier n'est pas atteinte faire : */
/*      - Stocker les mots contenus sur une ligne du fichier */
/*      dans les deux variables mot et trad */
/*      - Créer une cellule avec les deux chaînes de caractère */
/*      dans le dictionnaire en tant que mot à traduire et */
/*      mot traduit */
/*      - Insérer la nouvelle cellule dans la table */
/*      (sous-programme INSERT_CELL) */
/*      - Fermer le fichier */
/*      FIN */
/* */
/* Lexique */
/* */
/* En entrée : nom_fichier  Nom du fichier d'entrée (chaîne de caractères) */
/*      maj      Pointeur de tête de la table majeure */
/* */
/* Variables intermédiaires : */
/*      fichier      Pointeur sur le fichier d'entrée */
/*      lg_ligne      Entier représentant la longueur d'une ligne du fichier */
/*      ligne      Chaîne de caractère intermédiaire contenant la ligne courante */
/*      du fichier */
/*      mot, trad      Chaînes de caractères désignant respectivement le mot */
/*      à traduire et le mot traduit */
/*      nouv      Pointeur sur la nouvelle cellule à insérer */
/* */
/* En sortie : erreur      Entier valant 0 si le fichier est lu et 1 sinon */
/*-----*/

```

```

int Dictionnaire (char * nom_fichier, cellule_T ** maj)
{
    FILE * fichier;
    int erreur = 1;
    int lg_ligne;
    char ligne[60];
    char * mot, * trad;
    cellule_t * nouv;
    fichier = fopen(nom_fichier, "r");
    if (fichier != NULL)
    {
        INIT_TABLE(maj);
        erreur = 0;
        while(!feof(fichier))
        {
            fgets(ligne, 60, fichier);
            lg_ligne = strlen(ligne);
            if(lg_ligne >= 2)
            {
                if(ligne[lg_ligne-1] == '\n')
                {
                    lg_ligne = lg_ligne - 1;
                    ligne[lg_ligne] = '\0';
                }
                mot = strtok(ligne, ";");
                trad = strtok(NULL, "\\0");
                nouv = CREER_CELL(mot, trad);
                INSERT_CELL(nouv, *maj);
            }
            fclose(fichier);
        }
        return erreur;
    }
}

```

```

/*-----*/
/* TRADUCTION                Traduction d'un ensemble de mot                */
/*                                                                    */
/* Algorithme de principe:                                          */
/*      - Allouer un tableau de caractère traduction                */
/*      - Si l'allocation est réussie alors :                      */
/*          - Lire et stocker le premier mot de la chaîne d'entrée dans */
/*            une variable intermdiaire                             */
/*          - Tant que la fin de la chaîne de caractère d'entrée      */
/*            n'est pas atteinte faire :                             */
/*              - Rechercher la traduction de ce mot                 */
/*                (sous-programme RECHERCHE)                         */
/*              - Si le mot n'existe pas dans le dictionnaire alors : */
/*                - Stocker dans la variable intermédiaire trad le mot*/
/*                  dans la langue d'origine                         */
/*              - Ajouter le mot traduit à la fin de la chaîne de     */
/*                caractère traduction                                */
/*              - Lire et stocker le mot suivant dans une variable    */
/*                intermédiaire                                     */
/*                                                                    */
/*                                                                    FIN */
/*                                                                    */
/* Lexique                                                           */
/*                                                                    */
/* En entrée : maj          Adresse de la table majeure              */
/*               exp        Ensemble des mots à traduire (chaîne de caractères) */
/*                                                                    */
/* Variables intermédiaires :                                       */
/*      mot, trad          Chaînes de caractère intermédiaires contenant respectivement */
/*                          le mot à traduire et le mot traduit      */
/*      temp_exp           Chaîne de caractère temporaire contenant l'ensemble de mots */
/*                          en entrée                                */
/*                                                                    */
/* En sortie : traduction    Ensemble des mots traduits (chaîne de caractères) */
/*-----*/

```

```

char * TRADUCTION(cellule_T * maj, char * exp)
{
    char * mot, * trad, * temp_exp, * traduction;
    traduction = (char *)malloc(255*sizeof(char));
    temp_exp = (char *)malloc((strlen(exp)+1)*sizeof(char));
    if(traduction != NULL)
    {
        strcpy(traduction, "");
        strcpy(temp_exp, "");
        strcpy(temp_exp, exp);          /* Sauvegarde de exp car strtok tronque la chaîne */
        mot = strtok(temp_exp, " ");
        while(mot != NULL)
        {
            trad = RECHERCHE(maj, mot);
            if (trad == NULL)
            {
                trad = (char *)malloc((strlen(mot)+1)*sizeof(char));
                if (trad != NULL)
                    strcpy(trad, mot);
            }
            strcat(traduction, trad);
            free(trad);
            strcat(traduction, " ");
            mot = strtok(NULL, " ");
        }
    }
    free(temp_exp);
    return traduction;
}

```



```

/*-----*/
/*  COMPT_MOY          Retourne le nombre moyen de cellules contenues dans les  */
/*                      sous-tables                                           */
/*-----*/
/*  Algorithme de principe                                                    */
/*                      - Initialiser à zéro un compteur s et un flottant moy  */
/*                      - Pour chaque cellule de la table faire :              */
/*                        - Ajouter à s le contenu du compteur de la cellule    */
/*                      - Calcul de la moyenne                                */
/*                      FIN                                                    */
/*-----*/
/*  Lexique                                                                  */
/*-----*/
/*  En entrée : maj          Adresse de la table majeure                      */
/*-----*/
/*  Variables intermédiaires :                                              */
/*      s          Compteur du nombre de cellules dans la totalité des sous-tables de */
/*                  la table maj de taille HASH_MAX (entier)                  */
/*-----*/
/*  En sortie : moy          Flottant représentant le nombre moyen de cellules par sous-table */
/*-----*/

float COMPT_MOY(cellule_T * maj)
{
    int i;
    int s = 0;
    float moy = 0;
    for (i=0; i<HASH_MAX; i++)
    {
        s = s + (maj + i)->compteur;
    }
    moy = (float)s/HASH_MAX;
    return moy;
}

```

```

/*-----*/
/* SUPP_TABLE                Supprime une table                */
/*                                                                    */
/* Algorithme de principe:                                                                    */
/*      - Initialisation du pointeur précédent sur la première cellule */
/*      de la table majeure                                                                    */
/*      - Pour chacune des cellules de la table majeure faire : */
/*          - Tant que le pointeur précédent n'est pas NULL faire : */
/*              - Libérer la première cellule de la sous-table */
/*                  (sous-programme SUPP_CELL) */
/*      - Libérer la table majeure */
/*      FIN */
/*                                                                    */
/* Lexique                                                                    */
/* En entrée : prec                Pointeur sur le précédent de la cellule à supprimer */
/* Variables intermédiaires :                                                                    */
/*      cour                Pointeur sur la cellule à supprimer */
/*-----*/

void SUPP_TABLE(cellule_T * maj)
{
    int i;
    cellule_T * cour;
    for (i=0; i<HASH_MAX; i++)
    {
        cour = maj + i;
        if ((cour->pointeur) != NULL)
            SUPP_LCH(&(cour->pointeur));
    }
    free(maj);
}

```

2.5 Main.c

```
/*-----*/
/*          TP4 - Gestion d'un dictionnaire bilingue          */
/*                               Main.c                               */
/*-----*/

#include "Table.h"

int main()
{
    cellule_T * maj_vide = NULL, * maj_inex = NULL, * maj = NULL;
    char * trad_present = NULL, * trad_absent = NULL, * traduction1 = NULL, * traduction2 = NULL;
    char phrase1[] = "je pense donc je suis";
    char phrase2[] = "je suis une fleur";

    // Création des dictionnaires
    DICTIONNAIRE("vide.txt", &maj_vide);
    DICTIONNAIRE("inexistant.txt", &maj_inex);
    DICTIONNAIRE("anglais.txt", &maj);

    // Recherche de la traduction d'un mot
    trad_present = RECHERCHE(maj, "fleur");
    trad_absent = RECHERCHE(maj, "soleil");
    printf("Traduction de fleur: %s\n", trad_present);
    printf("Traduction de soleil: %s\n", trad_absent);
    free(trad_present); trad_present = NULL;
    free(trad_absent); trad_absent = NULL;

    // Traduction d'un ensemble de mots
    traduction1 = TRADUCTION(maj, phrase1);
    traduction2 = TRADUCTION(maj, phrase2);
    printf("Traduction de : %s -> %s \n", phrase1, traduction1);
    printf("Traduction de : %s -> %s \n", phrase2, traduction2);
    free(traduction1); traduction1 = NULL;
    free(traduction2); traduction2 = NULL;

    // Longueur moyenne des sous-tables du dictionnaire
    printf("Longueur moyenne pour le dictionnaire anglais.txt : %f\n", COMPT_MOY(maj));

    // Libération de la table
    SUPP_TABLE(maj); maj = NULL;
}
```

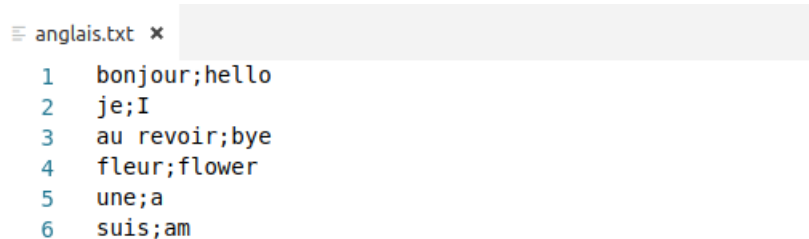
Partie 3

Compte rendu d'exécution

3.1 Jeux de test

Les mots à insérer dans le dictionnaire bilingue sont placés dans un fichier texte. Pour les jeux de tests, nous avons utilisé deux fichiers :

- anglais.txt (ci-dessous) contenant des mots français avec leur traduction en anglais ;
- vide.txt, le fichier vide.



```
anglais.txt x
1  bonjour;hello
2  je;I
3  au revoir;bye
4  fleur;flower
5  une;a
6  suis;am
```

Fichier utilisé pour les jeux de tests - Dictionnaire français :anglais

Les résultats des différents tests sont observés à travers le terminal et le débogueur ddd pour l'affichage de la table.

- Création d'un dictionnaire à partir d'un fichier vide
- Création d'un dictionnaire à partir d'un fichier inexistant
- Création d'un dictionnaire à partir d'un fichier non vide
- Recherche de la traduction d'un mot présent dans le dictionnaire
- Recherche de la traduction d'un mot absent du dictionnaire
- Traduction d'une phrase dont certains mots sont absents du dictionnaire
- Traduction d'une phrase dont tous les mots sont présents dans le dictionnaire
- Calcul de la longueur moyenne des sous-tables du dictionnaire
- Libération de la table

Les résultats des tests effectués sont affichés ci-après :

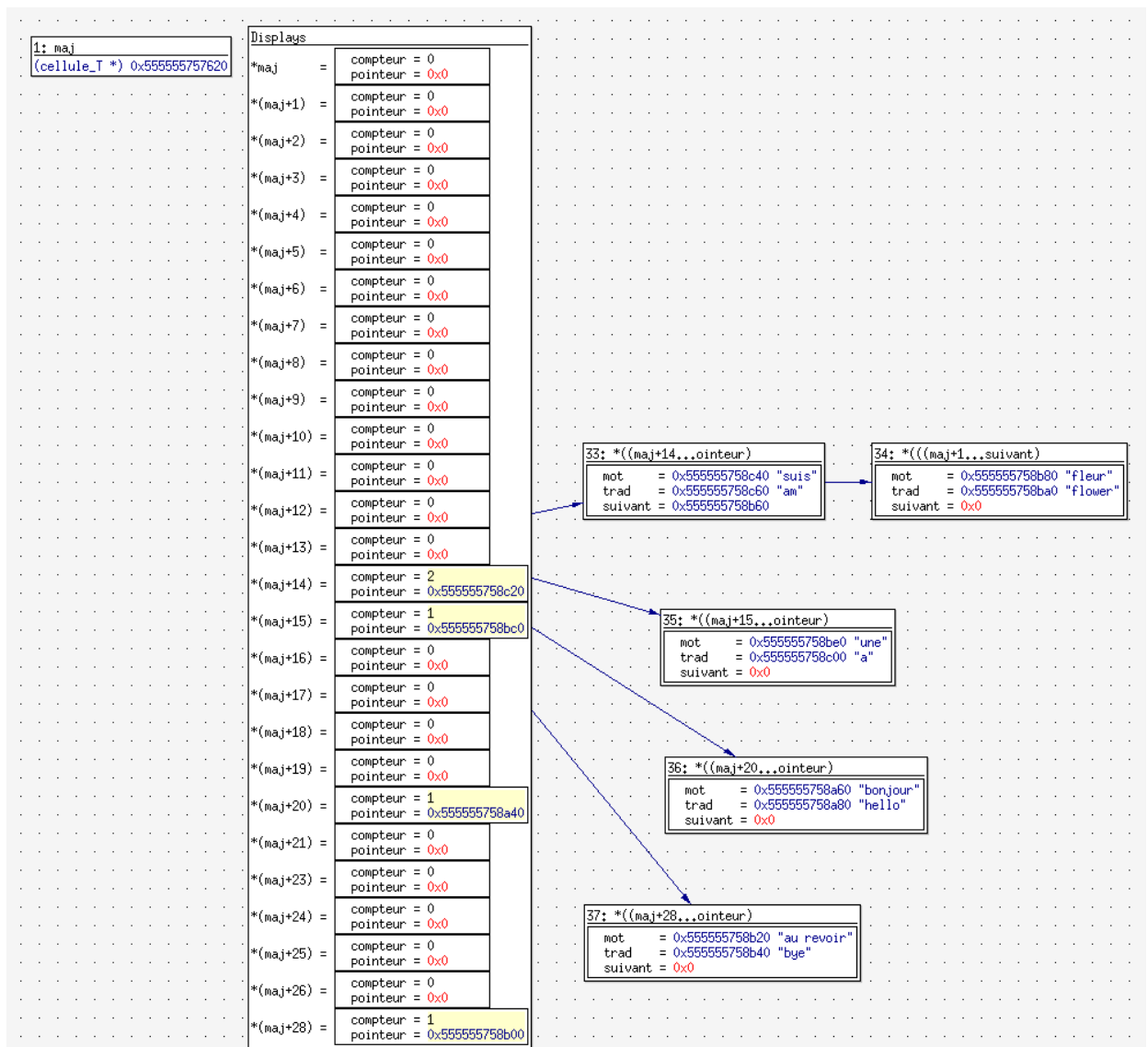
- Tests de création de dictionnaire à partir de :
Fichier vide

```
1: maj_vide
(cellule_T *) 0x0
```

Fichier inexistant

```
1: maj_inex
(cellule_T *) 0x0
```

Fichier complet



— Tests de recherche de la traduction de mots / de phrases :

```
Traduction de fleur: flower
Traduction de soleil: (null)
Traduction de : je pense donc je suis -> I pense donc I am
Traduction de : je suis une fleur -> I am a flower
```

— Tests du calcul de la longueur moyenne des sous-tables du dictionnaire :

```
Longueur moyenne pour le dictionnaire anglais.txt : 0.206897
```

— Test libération de l'arbre

```
1: maj
(cellule_T *) 0x0
```

Vérification avec Valgrind :

```
==7676== HEAP SUMMARY:
==7676==      in use at exit: 0 bytes in 0 blocks
==7676==    total heap usage: 38 allocs, 38 frees, 8,033 bytes allocated
==7676==
==7676== All heap blocks were freed -- no leaks are possible
==7676==
==7676== For counts of detected and suppressed errors, rerun with: -v
==7676== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2 Makefile

```
CC = gcc
OBSJ = Main.o Table.o Lch.o
ARGS = -Wextra -Wall -g
EXE = TP4

all : $(EXE)

$(EXE) : $(OBSJ)
    $(CC) -o $(EXE) $(OBSJ)
    @echo "Lancer le programme avec ./TP4"

%.o : %.c
    $(CC) -c $< $(ARGS)

clean :
    rm *.o
    rm TP4
```

À la compilation, la consigne suivante s'affiche sur le terminal :

```
nadoutchka@Nadoutchka:~/Téléchargements/TP4SDD$ make
gcc -c Main.c -Wextra -Wall -g
gcc -o TP4 Main.o Table.o Lch.o
Lancer le programme avec ./TP4
```