COMPTE RENDU DU TP4 DE SDD

Binôme :

AZEROUAL Mohammed

LINGOM NKAMGA David Cédric

Sommaire

1 Présentation générale

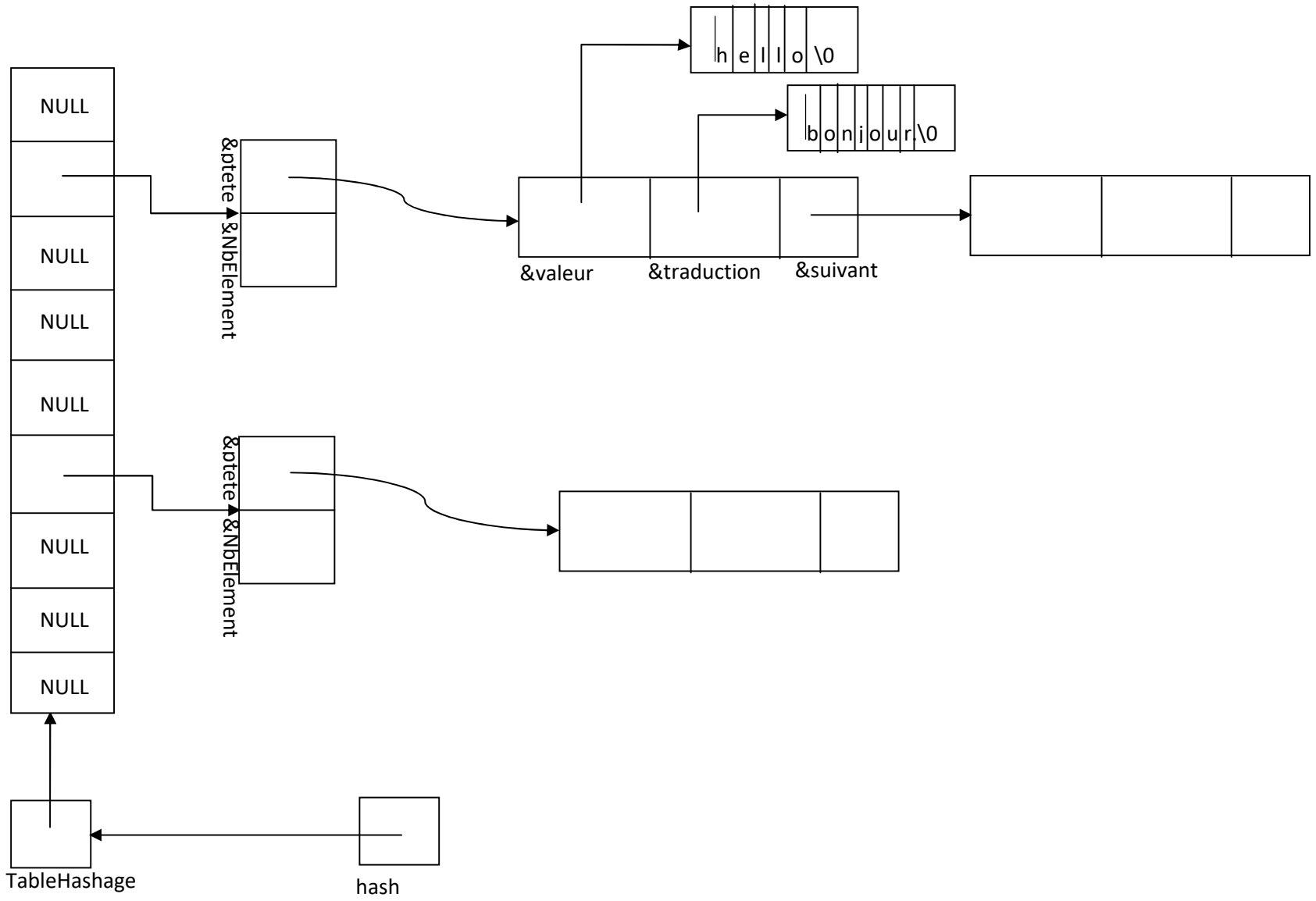
1.1 Description de l'objet du TP

Dans ce TP notre travail consiste à réaliser un gestionnaire d'application multilingue c'est-à-dire à partir d'un fichier texte contenant sur chaque ligne un mot et sa traduction dans un langage cible, séparé par un « ; ». C'est différentes lignes du fichier texte son lu et insérer dans une table hachage indirecte grâce à une fonction de hachage « hash_string » qui nous renvoie l'indice dans la table de hachage qui nous associe à la sous-table approprié. Cette ligne lue à partir du fichier texte est considérer comme une entrée sera recherchée dans la sous-table appropriée. Si elle n'existe pas on l'insère dans cette sous-table. Après insertion de toutes ses entrées du fichier texte, nous devons traduire un ensemble de mots dans un langage cible et ensuite donné la longueur moyenne des sous tables.

1.2 Description de la structure de données

Comme dis plus haut la structure de données utilisées pour cette réalisation est une table de hachage. En elle-même constitué d'un tableau statique de taille « HASH_MAX » avec chaque case qui est pointeur sur une structure de sous-table « mineur_t » qui contient deux champs. Le premier champ est un pointeur sur une sous-table. Cette sous-table en elle-même est une liste chainée dont les éléments sont de type « mot_t » qui est une structure de trois champs c'est-à-dire en premier nous avons un pointeur sur une chaine de caractère qui est un mot dans langage donné, en deuxième aussi pointeur sur un chaine de caractères mais qui représente la traduction du mot et le troisième champ pointeur sur l'élément suivant de la liste chainée. Ensuite comme deuxième champ de la structure « mineur_t » nous avons une variable de type « int » qui compte le nombre d'élément de la liste chainée.

1.3 Schéma de la table de hachage indirect



1.4 Description des fichiers des données utilisées (en entrée)

La lecture des données va être faite à partir des fichiers dictionnaire_1.txt, dictionnaire_2.txt, dictionnaire_3.txt, dictionnaire_4.txt, dictionnaire_5.txt et cela sera passé en ligne de commande. La création de la table sera visualiser grâce au DDD mais l'affiche de la longueur moyenne et de traduction d'une phrase sera ferra en sortie standard.

1.5 Organisation du code source

Notre code source constitué de fichier.c et fichier.h qui sont comme suit:

Nous avons le fichier principal « main.c » qui contient notre fonction principale qui fait appelle aux procédures : « LectureFichier » (qui lit les entrée à partir d'un fichier à être insérer dans la table). Sur cette sera effectuer des traitements grâce aux fonctions et procédures : « TraductionExpression » (traduit une expression en utilisant la table), «LongeurMoyenne» (calule la longeur moyenne des sous-tables) et « LibererTable » (libère la table majeure ainsi que les sous-tables). Notre fichier main.c contient comme fichier d'entête « hachage.h » qui est aussi inclus dans le fichier « hachage.c » et ce fichier d'entête contient des inclusions, des définitions de constantes symboliques et des prototypes comme suit :

```
/*------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                        */
/*                                                hachage.h                                                               */
/*                                                                                                                        */
/* Role : Déclarations des directives de préprocesseur, des constantes, des                                               */
/*          types personnels et des prototypes.                                                                           */
/*                                                                                                                        */
/*------------------------------------------------------------------------------------------------------------------------*/


#ifndef _GESTIONNAIRE_APPLICATION_MULTILINGUE_HACHAGE_H
#define _GESTIONNAIRE_APPLICATION_MULTILINGUE_HACHAGE_H



#include <ctype.h>
#include "./mot.h"
```

```
#define HASH_MAX 29



typedef struct mineur
{
        mot_t * ptete;
        unsigned int NbElement;
}mineur_t;
typedef mineur_t * table_t[HASH_MAX];



void IntialiseTableMajeure(table_t *);
void LibererSousTable(mineur_t **);
void LibererTable(table_t *);
void LectureFichier(char *, table_t *, enum bool *);
void InsertionChainee (mot_t **, mot_t *);
void SuppressionChainee(mot_t **);
unsigned int hash_string(const char *);
mot_t ** RechercheEntree(char *, enum bool *, table_t *, unsigned int);
void CreationTable(FILE *, table_t *, enum bool *, enum bool *);
float LongeurMoyenne(table_t *);
void TraductionMot(char *, table_t *);
void TraductionExpression(char *, table_t *);
```

Dans ce fichier d'entête est inclus les fichiers d'entêtes « ctype.h » et « mot.h ». Le fichier d'entête « mot.h » est aussi inclus dans « mot.c » et ce fichier d'entête contient des inclusions, des définitions de constantes symboliques, des définitions de type personnels et des prototypes comme suit :

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/*                                                mot.h                                                              */
/*                                                                                                                   */
/* Role : Déclarations des directives de préprocesseur, des constantes, des                                          */
/*          types personnels et des prototypes.                                                                      */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/


#ifndef _GESTIONNAIRE_APPLICATION_MULTILINGUE_MOT_H
#define _GESTIONNAIRE_APPLICATION_MULTILINGUE_MOT_H


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


#define TAILLECHAINE 40
```

```
enum bool{faux, vrai};

typedef struct mot

{

        char * valeur;

        char * traduction;

        struct mot * suivant;

}mot_t;



mot_t * AllocationMot();

void LectureLigneFichier(FILE *, char *, enum bool *);

int LongeurMot(char *, char);

void CopieMot(char *, char *, char);

mot_t * CreationMot(char *, enum bool *);

mot_t **  RecherchePrec (mot_t **, char *, enum bool *);



#endif
```

Dans ce fichier « mot.h » est aussi inclus les fichiers d'entête « stdio .h », « stdlib.h » et « string.h ».

2 Détail de chaque fonction

Au sujet des fonctions et procédures utilisés dans ce TP qui sont repartis dans des différents fichier.c, nous allons vous donner les détails des fonctions et procédures de chaque fichier.c.

Les fonctions et procédures du fichier « mot.c » sont détaillés comme suit:

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/*                                                mot.c                                                              */
/*                                                                                                                   */
/* Role : Définition des fonctions et procédures permettant la manipulation                                          */
/*          des mots.                                                                                                */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/
```

```
#include "./mot.h"
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/* AlloactionMot          Alloue le bloc devant contenir la valeur du mot et sa                                      */
/*                        traduction et un pointeur sur le suivant.                                                  */
/*                                                                                                                   */
/* En entrée          :          Rien en entrée                                                                      */
/*                                                                                                                   */
/* En sortie              : pmot - R'envoie l'adresse du bloc allouée.                                               */
/*                                                                                                                   */
/* Variable(s) locale(s) : pmot - Pointeur sur le bloc mot allouée.                                                  */
```

```
/*                                                                                                                                                    */
/*----------------------------------------------------------------------------------------------------------------------------------------------------*/


mot_t * AllocationMot()
{
        mot_t * pmot = (mot_t *)malloc(sizeof(mot_t));
        pmot->valeur = NULL;
        pmot->traduction = NULL;
        pmot->suivant = NULL;
        return pmot;
}


/*----------------------------------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                                                    */
/* LectureLigneFichier        Lit une ligne du fichier texte.                                                                                         */
/*                                                                                                                                                    */
/* En entrée          : f                  - Pointeur sur un fichier.                                                                             */
/*                      PcodeLecture - Pointeur sur une case contenant vrai si la                                                                 */
/*                                          lecture est c'est bien passée et faux si on                                                           */
/*                                          est en fin de lecture.                                                                                */
/*                                                                                                                                                    */
/* En sortie          : PcodeLecture - Pointeur sur une case contenant vrai si la                                                               */
/*                                          lecture est c'est bien passé et faux si on est                                                        */
/*                                          en fin de lecture.                                                                                    */
/* Variable(s) locale(s) : chaine     - Une chaine de caractères qui est la ligne lu.                                                            */
```

```
/*                                                                                      */
/*--------------------------------------------------------------------------------------*/


void LectureLigneFichier(FILE * f, char * chaine, enum bool * PcodeLecture)
{
	*PcodeLecture = vrai;
	if(!fgets(chaine, TAILLECHAINE, f))  /*si on est pas àla fin de la lecture*/
	{
		*PcodeLecture = faux;
	}
}


/*--------------------------------------------------------------------------------------*/
/*                                                                                      */
/* LongeurMot           Calcule la longueur d'un possédant un délimiteur.               */
/*                                                                                      */
/* En entrée          : chaine       - Pointeur sur une chaine de caractères.           */
/*                      delimiteur - Le délimiteur qui est un caractère.                */
/*                                                                                      */
/* En sortie          : longeur    - Retourne la longueur de ce mot délimiter.          */
/*                                                                                      */
/* Variable(s) locale(s) : i         - Variable de boucle.                              */
/*                                                                                      */
/*--------------------------------------------------------------------------------------*/
```

```
int LongeurMot(char * chaine, char delimiteur)
{
	static int i, longeur;
	i = 0;
	while(chaine[i] != delimiteur)		/*tantque l'on a pas atteint le délimiteur*/
	{
		++i;
	}
	longeur = i;
	return longeur;
}

/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/* CopieMot		Copie un mot délimiter d'une source vers une destination.                                    */
/*                                                                                                                   */
/* En entrée	: src		- Source du mot à copier.                                                    */
/*		  dest		- Destination du mot à copier.                                               */
/*		  delimiteur	- Délimiteur qui est un caractère.                                           */
/*                                                                                                                   */
/* En sortie	: dest		- Destination du mot à  copier.                                              */
/*                                                                                                                   */
/* Variable(s) locale(s) : i	- Variable de boucle.                                                        */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/
```

```
void CopieMot(char * dest, char * src, char delimiteur)
{
	static int i;
	i = 0;
	while(src[i] != delimiteur)	/*tantque l'on a pas atteint le délimiteur*/
	{
		dest[i] = src[i];
		++i;
	}
	dest[i] = '\0';
}


/*-------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                         */
/* CreationMot          Créer une cellule devant contenir un mot et sa traduction.                                        */
/*                                                                                                                         */
/* En entrée          : chaine            - Pointeur sur une chaine de caractères.                                        */
/*                      PcodeCreation     - Pointeur sur une case mémoire contenant                                       */
/*                                          vrai si la création c'est passé et faux sinon.                                 */
/*                                                                                                                         */
/* En sortie          : PcodeCreation     - Pointeur sur une case mémoire contenant                                       */
/*                                          vrai si la création c'est passé et faux sinon.                                 */
/*                      pmot              - Retourne l'adresse de la cellule contenant                                    */
/*                                          devant contenir le mot et ça traduction.                                       */
/*                                                                                                                         */
/* Variable(s) locale(s) : long           - Longeur d'une chaine de caractères.                                            */
```

```
/*                  pmot        - Pointeur sur la cellule contenant devant contenir le                  */
/*                                  le mot et ça traduction.                                            */
/*                                                                                                      */
/*------------------------------------------------------------------------------------------------------*/


mot_t * CreationMot(char * chaine, enum bool * PcodeCreation)
{
        static int longeur1, longeur2;
        mot_t * pmot = AllocationMot();
        longeur1 = 0;
        longeur2 = 0;
        *PcodeCreation = faux;          /*on suppose que la création pourrai mal se passer*/
        if(pmot)                    /*si alloué*/
        {
                longeur1 = LongeurMot(chaine, ';');
                pmot->valeur = (char *)malloc((longeur1 + 1) * sizeof(char));
                if(pmot->valeur)                        /*si l'alloctaion à marcher*/
                {
                        CopieMot(pmot->valeur, chaine, ';');        /*je copie le mot*/
                        longeur2 = strlen(&chaine[longeur1 + 1]);
                        pmot->traduction = (char *)malloc((longeur2) * sizeof(char));
                        if(pmot->traduction)                    /*si l'allocation à marcher*/
                        {
                                CopieMot(pmot->traduction, &chaine[longeur1 + 1], '\n');
                                *PcodeCreation = vrai;
                        }
```

```
                else
                {
                        printf("Erreur de création!\n");
                        free(pmot->valeur);
                        free(pmot);
                }
        }
        else
        {
                printf("Erreur de création!\n");
                free(pmot);
        }
    }
    return pmot;
}


/*-------------------------------------------------------------------------------------------------------------------*/
/* RechercheMot       Recherche un mot dans la liste chainée des mots.                                               */
/*                                                                                                                   */
/* En entrée:    PpteteListe - Pointeur de pointeur de tête de la liste chainée des                                  */
/*                             mots.                                                                                 */
/*               pvaleur     - Pointeur sur le mot à rechercher                                                    */
/*                             (qui est une chaine de caractères).                                                   */
/*               ptrouver    - booleen vrai si le mot est trouvé et faux sinon                                     */
/*                                                                                                                   */
/* En sortie:    prec    - Retourne l'adresse du pointeur de tête de liste                                           */
```

```
/*                          chainée des mots ou l'adresse de la case pointeur de          */
/*                          l'élément précédent dans la liste chainée de mots.            */
/*              ptrouver   - booleen vrai si le mot est trouvé et faux sinon.             */
/*                                                                                        */
/* Variable(s) locale(s):   pcour - Pointeur sur le mot courante.                         */
/*                          prec  - Pointeur de pointeur de tête de liste chainée des     */
/*                                  mots ou pointeur sur la case pointeur de l'élément    */
/*                                  précédent de la liste chainée des mots.               */
/*----------------------------------------------------------------------------------------*/



mot_t **  RecherchePrec (mot_t ** PpteteListe, char * pvaleur, enum bool * ptrouver)

{

        static mot_t * pcour, ** prec;

        pcour = *PpteteListe;

        prec = PpteteListe;

        *ptrouver = faux;

        while   ((pcour != NULL) && (strcmp(pvaleur, pcour->valeur) > 0)) /*Tantque je suis dans la liste et que ma chaine est plus grande*/

        {

                prec = &(pcour->suivant);                       /*On récupère l'adresse de la case pointeur de l'élément courant*/

                pcour = *prec;                                  /*Passe au suivant*/

        }

        if ((pcour != NULL) && (!strcmp(pvaleur, pcour->valeur)))       /*Si on trouve la chaine*/

        {

                *ptrouver = vrai;

        }
```

```
        return prec;
}
```

Les fonctionts et procédures procédures du fichier « hachage.c » sont comme suit :

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/*                                          hachage.c                                                                */
/*                                                                                                                   */
/* Role : Définition des fonctions et procédures permettant gestion de la table                                      */
/*        de hachage.                                                                                                */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/
```

```
#include "./hachage.h"
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/* InsertionChainee      Insère une nouvelle une cellule dans la liste chainée.                                      */
/*                                                                                                                   */
/* En entrée          : adpt - Pointeur de pointeur de tête de liste chainée ou                                      */
/*                             pointeur sur la case pointeur de l'élément précédent.                                 */
/*                      pmot - Pointeur sur l'élément à insérer.                                                     */
/*                                                                                                                   */
/* En sortie          : adpt - Pointeur de pointeur de tête de liste chainée ou                                      */
/*                             pointeur sur la case pointeur de l'élément précédent.                                 */
```

```
/*                                                                                                                    */
/* Variable(s) locale(s) :     Rien en variable locale                                                                */
/*                                                                                                                    */
/*--------------------------------------------------------------------------------------------------------------------*/


void InsertionChainee (mot_t ** adpt, mot_t * pmot)
{
        pmot->suivant = *adpt;
        *adpt = pmot;
}


/*--------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                    */
/* SuppressionChainee         Supprime un bloc                                                                        */
/*                            (Dans notre cas contenant un mot et ça traduction).                                     */
/*                                                                                                                    */
/* En entrée          : adpt  - Pointeur de pointeur de tête de liste chainée ou pointeur                             */
/*                              sur la case pointeur de l'élément précédent de la liste                               */
/*                              chainée.                                                                              */
/*                                                                                                                    */
/* En sortie          : adpt   - Pointeur de pointeur de tête de liste chainée ou pointeur                            */
/*                              sur la case pointeur de l'élément précédent de la liste                               */
/*                             chainée.                                                                               */
/* Variable(s) locale(s) : pcour - Pointeur sur l'élément à supprimer.                                                */
/*                                                                                                                    */
```

```
/*-------------------------------------------------------------------------------------------------------------------------*/



void SuppressionChainee(mot_t ** adpt)
{
        mot_t * pcour = *adpt;     /*recupère l'adresse de l'élément à supprimer*/
        *adpt = pcour->suivant;    /*pointe sur l'élément après l'élément à supprimer*/
        free(pcour);               /*supprime l'élément*/
        pcour = NULL;
}



/*-------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                         */
/* hash_string          Fonction de hachage qui étant donnée une chaine                                                    */
/*                      considérer comme la clé calcule et nous retourne l'indice                                          */
/*                      de la case au niveau de la table majeur devant pointer                                             */
/*                      sur la bonne table mineur approprié.                                                               */
/*                                                                                                                         */
/* En entrée          : str  - Pointeur sur une chaine de caractères.                                                      */
/*                                                                                                                         */
/* En sortie          : hash - L'indice de la table majeur associé à la clé.                                               */
/*                                                                                                                         */
/* Variable(s) locale(s) : hash - Retourne L'indice de la table majeur associé à la                                        */
/*                                clé.                                                                                     */
/*                         s      - Pointeur sur une chaine de caractères.                                                 */
/*                                                                                                                         */
```

```
/*-------------------------------------------------------------------------------------------------------------------------*/


unsigned int hash_string(const char * str)
{
        static unsigned int hash;           /*  fonction de hachage de D.J. Bernstein*/
        const char *s;
        hash = 5381;
        for (s = str; *s; s++)
        {
                hash = ((hash << 5) + hash) + tolower(*s);
        }
        return (hash & 0x7FFFFFFF)% HASH_MAX;
}


/*-------------------------------------------------------------------------------------------------------------------------*/
/* RechercheEntree          Recherche un mot dans le dictionnaire.                                                         */
/*                                                                                                                         */
/* En entrée          : hash      - Pointeur de pointeur sur un tableau de pointeurs de tables                            */
/*                                  mineurs (liste chainées).                                                              */
/*                      pvaleur    - Pointeur sur le mot à rechercher (chaine de caractères).                              */
/*                      ptrouver   - Booléen valant vrai si le mot est trouvé faux sinon.                                  */
/*                      IndiceHash - L'indice de la table majeur dont le contenu de la case est un                         */
/*                                   pointeur sur une liste chainée censé contenir le bloc du mot                          */
/*                                   rechercher.                                                                           */
/*                                                                                                                         */
```

```
/* En sortie          : prec      - Retourne l'adresse du pointeur de tête de liste chainée des mots       */
/*                                  ou l'adresse de la case pointeur de l'élément précédent dans la       */
/*                                  liste chainée de mots.                                                */
/*                      ptrouver   - Booléen valant vrai si le mot est trouvé faux sinon.                 */
/*                                                                                                        */
/* Variable(s) locale(s) : Ppteteliste - Pointeur de pointeur de tête de liste chainée des mots.         */
/*--------------------------------------------------------------------------------------------------------*/



mot_t ** RechercheEntree(char * pvaleur, enum bool * ptrouver, table_t * hash, unsigned int IndiceHash)
{
        static mot_t ** Ppteteliste, ** prec;
        if(!(*hash)[IndiceHash])            /*si la structure représentant la table mineur n'existe pas*/
        {
                (*hash)[IndiceHash] = (mineur_t *)malloc(sizeof(mineur_t));
                if((*hash)[IndiceHash])             /*si l'allocation de la structure à marcher*/
                {
                        (*hash)[IndiceHash]->ptete = NULL;
                        (*hash)[IndiceHash]->NbElement = 0;
                }
        }
        Ppteteliste = &((*hash)[IndiceHash]->ptete);
        prec = RecherchePrec(Ppteteliste, pvaleur, ptrouver);
        return prec;
}
/*--------------------------------------------------------------------------------------------------------*/
/*                                                                                                        */
```

```
/* CreationTable        Crée les différentes tables mineures avec leurs éléments si nécessaire.        */
/*                                                                                                     */
/* En entrée          : hash              - Pointeur de pointeur sur un tableau de pointeurs de tables    */
/*                                          mineurs (liste chainées).                                  */
/*                      PcodeLecture      - Contient vrai si la lecture d'une ligne du fichier c'est     */
/*                                          bien passé et faux sinon.                                  */
/*                      PcodeCreation     - Contient vrai si la création des éléments des               */
/*                                          différentes table mineurs se passe bien et faux sinon.     */
/*                                                                                                     */
/* En sortie          : hash      - Pointeur de pointeur sur un tableau de pointeurs de tables        */
/*                                  mineurs (liste chainées).                                          */
/*                      PcodeLecture   - Contient vrai si la lecture d'une ligne de fichier c'est      */
/*                                       bien passé et faux sinon.                                     */
/*                      PcodeCreation  - Contient vrai si la création des éléments des                */
/*                                       différentes table mineurs se passe bien et faux sinon.        */
/*                                                                                                     */
/* Variable(s) locale(s) : IndiceHash   - Indice du tableau renvoyer par la fonction de hachage       */
/*                         trouver      - Contient vrai si une entrée existe déjà et faux sinon       */
/*                         chaine       - Pointeur sur une chaine de caractère.                       */
/*                         prec         - Pointeur de pointeur de tête de liste chainée ou un         */
/*                                        pointeur de la case pointeur de l'élément précédent.        */
/*                                                                                                     */
/*-----------------------------------------------------------------------------------------------------*/
```

```
void CreationTable(FILE * f, table_t * hash, enum bool * PcodeLecture, enum bool * PcodeCreation)
{
	unsigned int IndiceHash;
	enum bool trouver;
	char * chaine = (char *)malloc(TAILLECHAINE * sizeof(char));
	mot_t ** prec = NULL, * pmot = NULL;
	*PcodeLecture = vrai;
	*PcodeCreation = vrai;
	if(chaine)                                          /*si allocation de la chaine*/
	{
		do
		{
			LectureLigneFichier(f, chaine, PcodeLecture);
			if(*PcodeLecture)
			{
				pmot = CreationMot(chaine, PcodeCreation);
				if(*PcodeCreation)                  /* si la création à reussi*/
				{
					IndiceHash = hash_string(pmot->valeur);
					prec = RechercheEntree(pmot->valeur, &trouver, hash,
IndiceHash);
					if(!trouver)
					{
						InsertionChainee(prec, pmot);
						((*hash)[IndiceHash])->NbElement++;
					}
					else
					{
```

```
                                        free(pmot->valeur);
                                        free(pmot->traduction);
                                        SuppressionChainee(&pmot);
                                    }
                                }
                            }
                    }while(*PcodeLecture && *PcodeCreation);                    /*tantque je ne suis pas à la fin de la lecture*/
                    free(chaine);
            }
            else
            {
                    *PcodeCreation = faux;
            }
}


/*-------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                         */
/* InitialiseTableMajeure  Initialise les cases du tableau à NULL.                                                         */
/*                                                                                                                         */
/* En entrée          : hash      - Pointeur de pointeur sur un tableau de pointeurs de tables                         */
/*                                  mineurs (liste chainées).                                                              */
/*                                                                                                                         */
/* En sortie          : hash      - Pointeur de pointeur sur un tableau de pointeurs de tables                         */
/*                                  mineurs (liste chainées).                                                              */
/*                                                                                                                         */
/* Variable(s) locale(s) : i   - Variable de boucle.                                                                       */
```

```
/*                                                                                    */
/*------------------------------------------------------------------------------------*/


void IntialiseTableMajeure(table_t * hash)
{
	int i;
	for(i = 0; i < HASH_MAX; ++i)
	{
		(*hash)[i] = NULL;
	}
}


/*------------------------------------------------------------------------------------*/
/*                                                                                    */
/* LibererSousTable       Libère la sous table qui ici est une liste chainée.         */
/*                                                                                    */
/* En entrée            : SousTable - Pointeur de pointeur sur une structure de table mineur   */
/*                                    (liste chainée).                                */
/*                                                                                    */
/* En sortie            :             Rien en sortie.                                 */
/*                                                                                    */
/* Variable(s) locale(s) : pmot      - Pointeur sur une structure contenant un mot et sa    */
/*                                      traduction.                                   */
/*                                                                                    */
/*------------------------------------------------------------------------------------*/
```

```
void LibererSousTable(mineur_t ** SousTable)
{
	mot_t * pmot = NULL;
	while(((*SousTable) != NULL) && ((*SousTable)->ptete != NULL))  /*tantqu'il existe une sous table et qu'elle n'est pas libérer*/
	{
		pmot = (*SousTable)->ptete;		/*Pointe sur le mot courant*/
		free(pmot->valeur);
		free(pmot->traduction);
		SuppressionChainee(&((*SousTable)->ptete));  /*supprime l'élément en tête de la liste chainée courante*/
	}
	if((*SousTable) != NULL)
	{
		free(*SousTable);			/*libÃ¨re la structure de table mineur*/
	}
}
```

```
/*-------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                         */
/* LibererTable		Libère les différentes sous table à partir de la table majeur.                                     */
/*                                                                                                                         */
/* En entrée		: hash	- Pointeur de pointeur sur un tableau de pointeurs de tables                               */
/*                                  mineurs (liste chainées).                                                              */
/*                                                                                                                         */
/* En sortie		: hash	- Pointeur de pointeur sur un tableau de pointeurs de tables                               */
/*                                  mineurs (liste chainées).                                                              */
```

```
*/
/*                                                                                                                    */
/* Variable(s) locale(s) : i    - Variable de boucle.                                                                 */
/*                                                                                                                    */
/*--------------------------------------------------------------------------------------------------------------------*/


void LibererTable(table_t * hash)
{
        int i = 0;
        for(i = 0; i < HASH_MAX; ++i)
        {
                LibererSousTable(&(*hash)[i]);
        }
}


/*--------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                    */
/* LectureFichier        Permet d'ouvrir un fichier, créer la table de hachage indirect et                            */
/*                       d'insérer chaque ligne du fichier dans la sous table appropriée.                             */
/*                                                                                                                    */
/* En entrée          : NomFichier     - Pointeur sur chaine de caractères.                                          */
/*                      hash           - Pointeur de pointeur sur un tableau de pointeurs de tables                  */
/*                                       mineurs (liste chainées).                                                    */
/*                      PcodeLecture - Pointeur sur une case mémoire contenant vrai si la                           */
/*                                     lecture c'est bien passée et faux sinon.                                       */
```

```
/*                                                                                                                    */
/* En sortie          : PcodeLecture - Pointeur sur une case mémoire contenant vrai si la                             */
/*                                     lecture c'est bien passée et faux sinon.                                       */
/*                      hash          - Pointeur de pointeur sur un tableau de pointeurs de tables                    */
/*                                     mineurs (liste chainées).                                                      */
/*                                                                                                                    */
/* Variable(s) locale(s) : CodeCreation - Pointeur sur une case mémoire contenant vrai si la                          */
/*                                        création de la table c'est bien passée et faux sinon.                       */
/*                          f           - Pointeur sur un fichier.                                                    */
/*                                                                                                                    */
/*--------------------------------------------------------------------------------------------------------------------*/


void LectureFichier(char * NomFichier, table_t * hash, enum bool * PcodeLecture)
{
        FILE * f = fopen(NomFichier, "r");
        enum bool CodeCreation;
        if(f)                                   /*si l'ouverture du fichier à réussi*/
        {
                CreationTable(f, hash, PcodeLecture, &CodeCreation);
                if(!CodeCreation)                       /*si erreur dans la création de table*/
                {
                        LibererTable(hash);
                        *PcodeLecture = CodeCreation;
                }
                else
                {
```

```
                        *PcodeLecture = vrai;                   /*la lecture c'est bien passé*/
                }
                fclose(f);
        }
}


/*-------------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                         */
/* LongeurMoyenne       Calcule la longueur moyenne des sous-tables.                                                       */
/*                                                                                                                         */
/* En entrée          : hash             - Pointeur de pointeur sur un tableau de pointeurs de tables                     */
/*                                         mineurs (liste chainées).                                                       */
/*                                                                                                                         */
/* En sortie          : moy  - Retourne la longueur moyenne des sous-tables.                                             */
/*                                                                                                                         */
/* Variable(s) locale(s) : i          - Variable de boucle.                                                                */
/*                         longeur - La somme des toutes les longueurs.                                                    */
/*                         moy     - La longueur moyenne des sous-tables.                                                  */
/*                                                                                                                         */
/*-------------------------------------------------------------------------------------------------------------------------*/
```

```
float LongeurMoyenne(table_t * hash)
{
	int i, longeur = 0;
	float moy = 0;
	for(i = 0; i < HASH_MAX; ++i)
	{
		if((*hash)[i])		/*si la table mineur existe*/
		{
			longeur += (*hash)[i]->NbElement;
		}
	}
	moy = (longeur * 1.0)/HASH_MAX;
	return moy;
}


/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/* TraductionMot        Traduit un mot d'une langue à une autre et l'affiche.                                        */
/*                                                                                                                   */
/* En entrée          : MotTraduire   - Pointeur sur chaine de caractère qui représente le mot                      */
/*                                      à traduire.                                                                  */
/*                      hash          - Pointeur de pointeur sur un tableau de pointeurs de tables                 */
/*                                      mineurs (liste chainées).                                                    */
/*                                                                                                                   */
/* En sortie          :                 Rien en sortie.                                                            */
/*                                                                                                                   */
```

```
/* Variable(s) locale(s) : trouver          - Variable contenant vrai si on a trouver l'entrée          */
/*                                            recherché et faux sinon.                                  */
/*                         IndiceHachage - C'est un indice du tableau de hachage calculé par la       */
/*                                            fonction de hachage.                                      */
/*                         prec             - Pointeur de pointeur sur un mot.                        */
/*                                                                                                      */
/*--------------------------------------------------------------------------------------------------------*/



void TraductionMot(char * MotATraduire, table_t * hash)
{
        static enum bool trouver;
        static unsigned int IndiceHachage;
        static mot_t ** prec;
        trouver = faux;
        IndiceHachage = hash_string(MotATraduire);
        prec = RechercheEntree(MotATraduire, &trouver, hash, IndiceHachage);
        if(trouver)
        {
                printf("%s ",(*prec)->traduction);
        }
        else
        {
                Printf("%s ", MotATraduire);
        }
}
```

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/* TraductionExpression    Traduit une expression donnée dans un langage cible.                                      */
/*                                                                                                                   */
/* En entrée          : expression - Pointeur sur une chaine de caractères.                                          */
/*                      hash       - Pointeur de pointeur sur un tableau de pointeurs de tables                   */
/*                                   mineurs (liste chainées).                                                       */
/*                                                                                                                   */
/* En sortie          :            Rien en sortie.                                                                   */
/*                                                                                                                   */
/* Variable(s) locale(s) : ch      - Pointeur de chaine de caractères.                                               */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/


void TraductionExpression(char * expression, table_t * hash)
{
        char * ch = strtok(expression, " ");
        while(ch != NULL)
        {
                TraductionMot(ch, hash);
                ch = strtok(NULL, " ");
        }
        printf("\n");
}
```

Le fichier « main.c » qui contient notre fonction principal faisant appel autres fonctions citez plu haut est comme suit:

```
/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/*                                              main.c                                                               */
/*                                                                                                                   */
/* Role : Contient notre fonction principale main qui fait appelle d'autres fonctions et procédures                  */
/*        tels que: LectureFichier, Traduction et LongeurMoyenne permettant des traiments.                           */
/*        sur des mots.                                                                                              */
/*                                                                                                                   */
/*-------------------------------------------------------------------------------------------------------------------*/


#include "./hachage.h"


/*-------------------------------------------------------------------------------------------------------------------*/
/*                                                                                                                   */
/* main             Fonction principale faisant appelle à des fonctions et procédures qui                            */
/*                  permettent de lire des mots fichier ainsi que leur traduction et                                 */
/*                  de stocker dans des tables qui seront ensuite utiliser pour faire des                            */
/*                  traductions.                                                                                     */
/*                                                                                                                   */
/*                                                                                                                   */
/* En entrée        : argc      - Compte de le nombre d'arguments passés en ligne de                                 */
/*                                commande.                                                                          */
/*                    argv      - Pointeur de pointeur de  chaines caractères qui sont                               */
/*                                rien d'autres que les arguments passés en ligne de                                 */
```

```
/*                              commande.                                                                    */
/*                                                                                                           */
/* En sortie        :          Rien en sortie.                                                              */
/*                                                                                                           */
/* Variable(s) locale(s) : CodeLecture  - Variable contenant vrai si la lecture à partir du                  */
/*                                        fichier c'est passé et faux sinon.                                 */
/*                         TableHashage - Un tableau de pointeur si des tables mineurs.                      */
/*                         phrase       - Un tableau de caractères qui constitue la phrase à                 */
/*                                        traduire.                                                          */
/*                                                                                                           */
/*-----------------------------------------------------------------------------------------------------------*/


int main(int argc, char ** argv)
{
        enum bool CodeLecture;
        table_t TableHashage;
        char phrase[80];
        if(argc == 2)
        {
                IntialiseTableMajeure(&TableHashage);
                LectureFichier(argv[1], &TableHashage, &CodeLecture);
                if(CodeLecture)                         /*si la lecture c'est bien passée*/
                {
                        strcpy(phrase, "We often play football on the pitch just behind our school");
                        printf("La trauction de la phrase:\n%s\n", phrase);
                        TraductionExpression(phrase, &TableHashage);
```

```
                printf("La longueur moyenne des sous-tables est: %f\n", LongeurMoyenne(&TableHashage));
                LibererTable(&TableHashage);
            }
            else
            {
                printf("Erreur dans la lecture du fichier");
            }
        }
        else
        {
            printf("Erreur vous n'avez pas entrer le bon nombre d'arguments!\n");
        }
        return 0;
}
```

3 Compte rendu d'exécution

3.1 Makefile

```
#compilateur
CC = gcc



#Les options
CFLAGS = -ansi -pedantic -Wall -Wextra -g -O2
LDFALGS = -lm



#Executable
EXEC = main


#Liste des fichiers objets
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)



#Règle d'exécution
all: $(EXEC)



#Règle de production de l'exécutable
main: $(OBJ)
    $(CC) -o $@ $^ $(LDFLAGS)




#Règle de production des fichiers objets
%.o: %.c
    $(CC) -c $< $(CFLAGS)



#Règle de nettoyage

clean:
    rm $(OBJ)

mrpropre: clean
    rm $(EXEC)
```

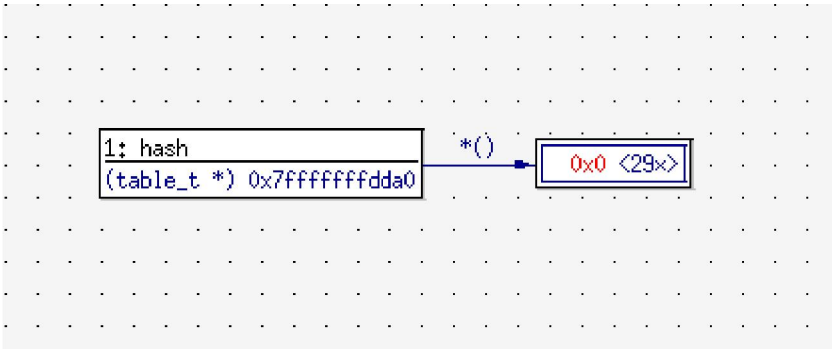
3.2 Jeux de test complets

3.2.1 Création de la table majeur ainsi les sous tables appropriés

3.2.1.1 Lorsque le fichier est vide

- Dans ce cas particulier nous avons utilisé le fichier dictionnaire_1.txt comme donnée d'entrée.

- **Résultat de l'exécution avec DDD**



- **Résultat de l'exécution sur le terminal**

```
cedric@cedric-X302LA:~/TP4_3SDD$ main dictionnaire_1.txt
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.000000
```

- **Exécution avec valgrind**

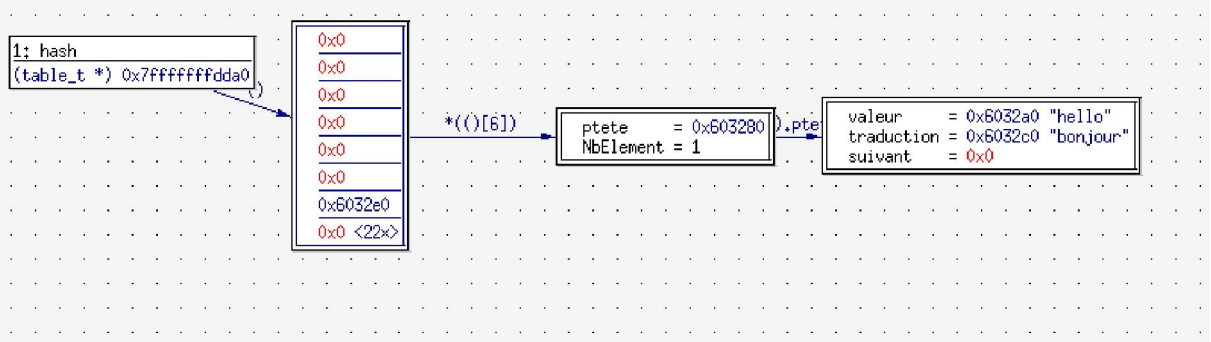
```
cedric@cedric-X302LA:~/TP4_3SDD$ valgrind main dictionnaire_1.txt
==2721== Memcheck, a memory error detector
==2721== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2721== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2721== Command: main dictionnaire_1.txt
==2721==
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.000000
==2721==
==2721== HEAP SUMMARY:
==2721==     in use at exit: 0 bytes in 0 blocks
==2721==   total heap usage: 2 allocs, 2 frees, 608 bytes allocated
==2721==
==2721== All heap blocks were freed -- no leaks are possible
==2721==
==2721== For counts of detected and suppressed errors, rerun with: -v
==2721== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2.1.2 Lorsque le fichier ne contient qu'un mot

- **Pour cas la donnée entrée est dans le fichier dictionnaire_2.txt représenter par cette capture d'écran:**

```
hello;bonjour
```

- **Résultat de l'exécution avec DDD**



- **Résultat de l'exécution sur le terminal**

```
cedric@cedric-X302LA:~/TP4_3SDD$ main dictionnaire_2.txt
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.034483
```

- **Exécution avec valgrind**

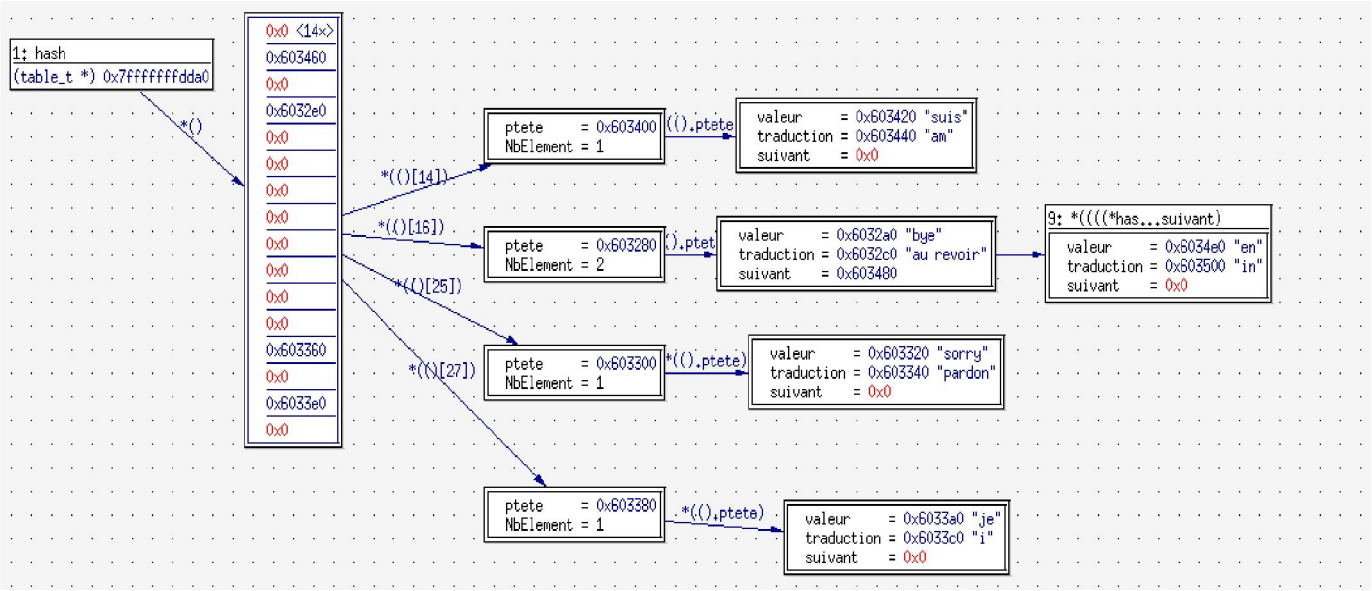
```
cedric@cedric-X302LA:~/TP4_3SDD$ valgrind main dictionnaire_2.txt
==2933== Memcheck, a memory error detector
==2933== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2933== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2933== Command: main dictionnaire_2.txt
==2933==
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.034483
==2933==
==2933== HEAP SUMMARY:
==2933==     in use at exit: 0 bytes in 0 blocks
==2933==   total heap usage: 6 allocs, 6 frees, 662 bytes allocated
==2933==
==2933== All heap blocks were freed -- no leaks are possible
==2933==
==2933== For counts of detected and suppressed errors, rerun with: -v
==2933== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2.1.3 Lorsque le fichier contient au moins deux avec certains qui sont répétées

- **Les données d'entrées sont dans le fichier** dictionnaire_3.txt **comme suit:**

```
bye;au revoir
sorry;pardon
je;i
suis;am
bye;au revoir
en;in
sorry;pardon
```

- **Résultat de l'exécution avec DDD**



- **Résultat de l'exécution sur le terminal**

```
cedric@cedric-X302LA:~/TP4_3SDD$ main dictionnaire_3.txt
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.172414
```

- **Exécution avec valgrind**

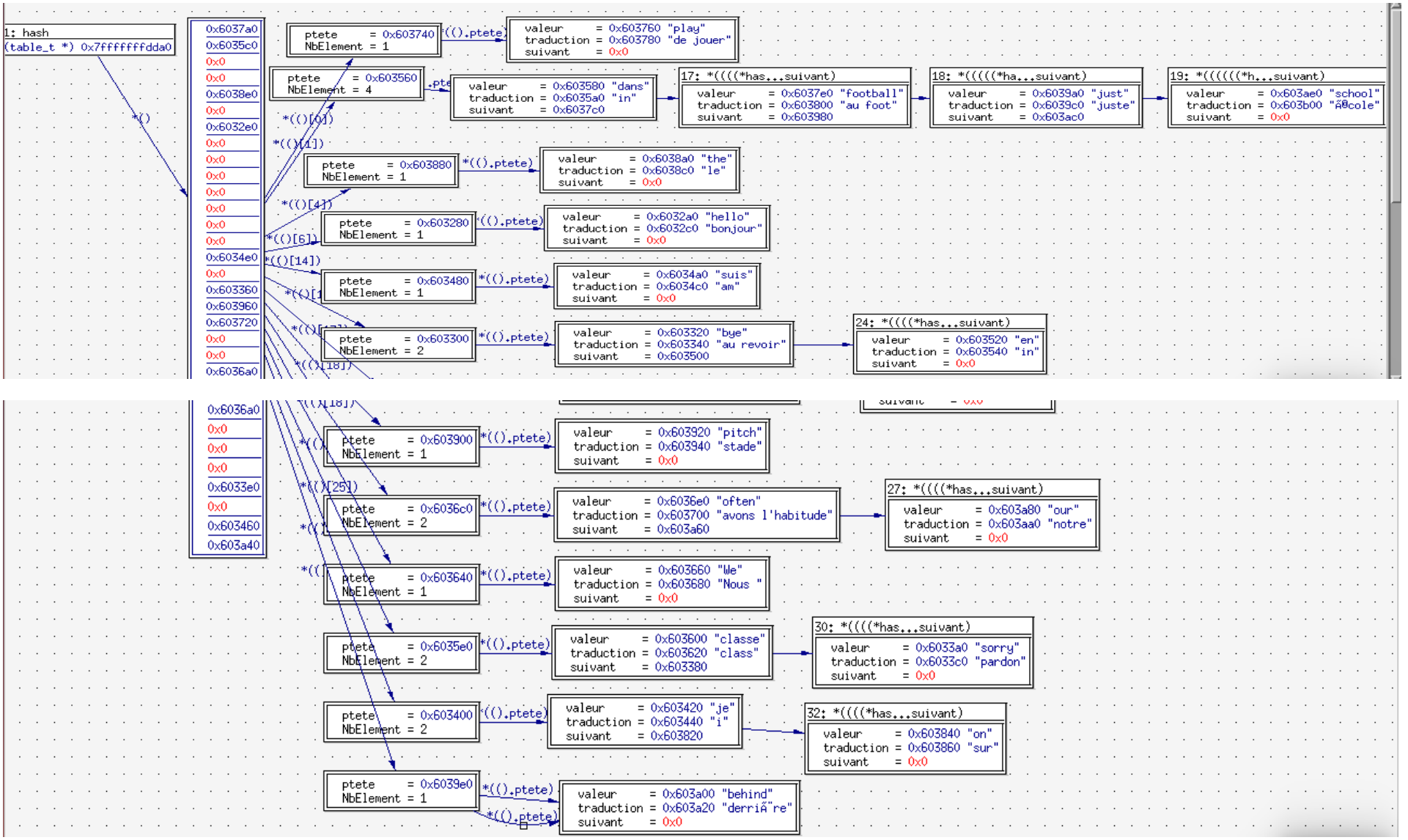
```
cedric@cedric-X302LA:~/TP4_3SDD$ valgrind main dictionnaire_3.txt
==3211== Memcheck, a memory error detector
==3211== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3211== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3211== Command: main dictionnaire_3.txt
==3211==
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.172414
==3211==
==3211== HEAP SUMMARY:
==3211==     in use at exit: 0 bytes in 0 blocks
==3211==   total heap usage: 27 allocs, 27 frees, 913 bytes allocated
==3211==
==3211== All heap blocks were freed -- no leaks are possible
==3211==
==3211== For counts of detected and suppressed errors, rerun with: -v
==3211== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2.1.4 Lorsque fichier contient au moins deux mots tous distincts

- **Les données d'entrées sont dans le fichier** dictionnaire_4.txt

```
hello;bonjour
bye;au revoir
sorry;pardon
je;i
suis;am
en;in
dans;in
classe;class
We;Nous
often;avons l'habitude
play;de jouer
football;au foot
on;sur
the;le
pitch;stade
just;juste
behind;derrière
our;notre
school;école
```

- **Résultat de l'exécution avec DDD**



- **Résultat de l'exécution sur le terminal**

```
cedric@cedric-X302LA:~/TP4_3SDD$ main dictionnaire_4.txt
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.655172
```

- **Exécution avec valgrind**

```
cedric@cedric-X302LA:~/TP4_3SDD$ valgrind main dictionnaire_4.txt
==3380== Memcheck, a memory error detector
==3380== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3380== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3380== Command: main dictionnaire_4.txt
==3380==
1. Creation de table.
2. Traduction de la phrase.
1
La longueur moyenne des sous-tables est: 0.655172
==3380==
==3380== HEAP SUMMARY:
==3380==     in use at exit: 0 bytes in 0 blocks
==3380==   total heap usage: 71 allocs, 71 frees, 1,478 bytes allocated
==3380==
==3380== All heap blocks were freed -- no leaks are possible
==3380==
==3380== For counts of detected and suppressed errors, rerun with: -v
==3380== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2.2 Traduction d'une phrase

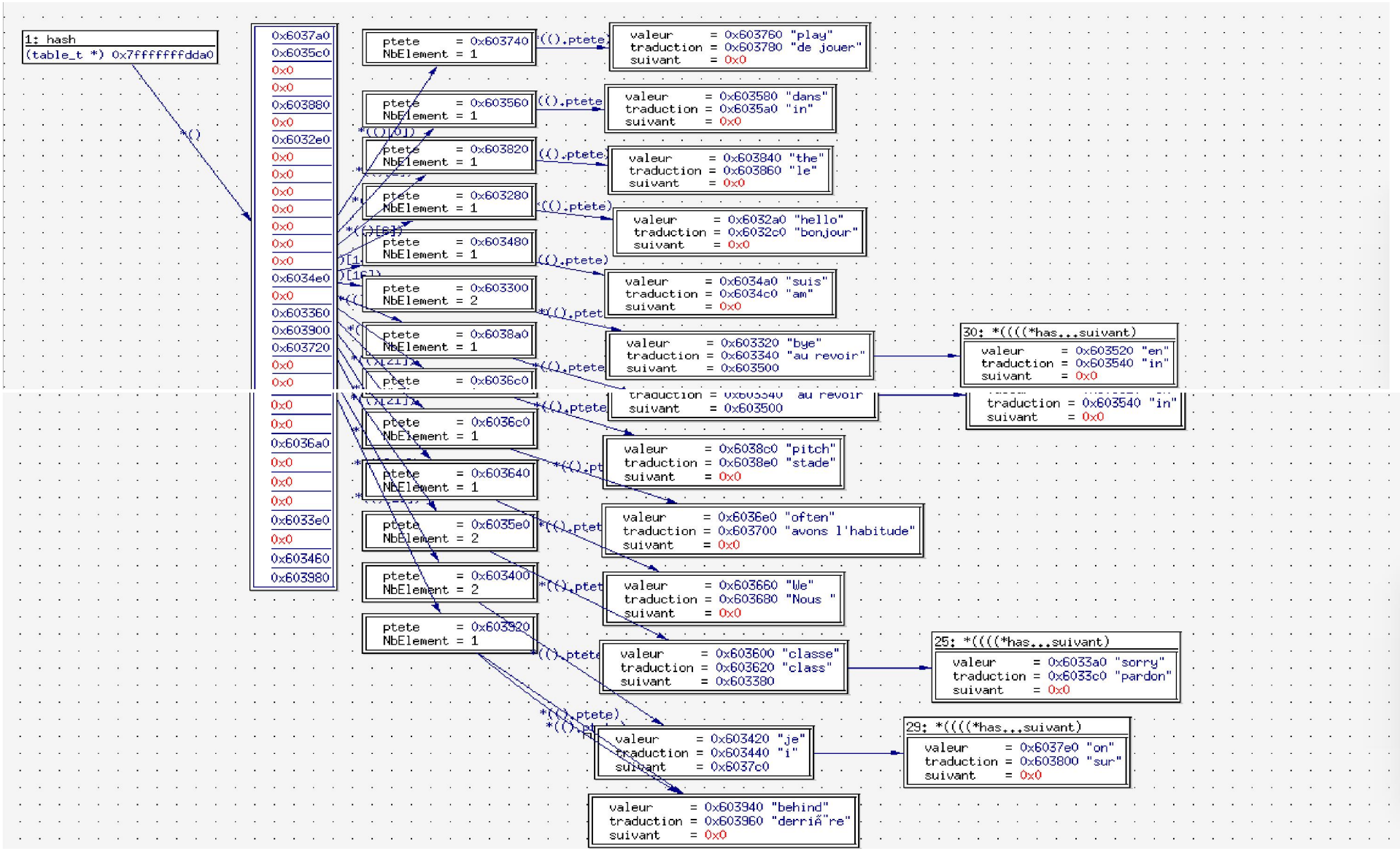
Pour les différents cas particuliers concernant la traduction d'une phrase de l'Anglais au français nous utiliserons les données contenues dans dictionnaire_4.txt et dictionnaire_5.txt.

3.2.2.1 Lorsque certains mots de la phrase n'existe pas dans la table

- **Pour ce cas particulier nous aurons besoins du fichier dictionnaire_5.txt dont son contenu est comme suit :**

```
hello;bonjour
bye;au revoir
sorry;pardon
je;i
suis;am
en;in
dans;in
classe;class
We;Nous
often;avons l'habitude
play;de jouer
on;sur
the;le
pitch;stade
behind;derrière
```

- **Résultat de l'exécution avec DDD**



- **Résultat de l'exécution sur le terminal**

```
cedric@cedric-X302LA:~/TP4_3SDD$ main dictionnaire_5.txt
1. Creation de table.
2. Traduction de la phrase.
2
La trauction de la phrase:
We often play football on the pitch just behind our school
Nous  avons l'habitude de jouer football sur le stade just derrière our school
```

- **Exécution avec valgrind**

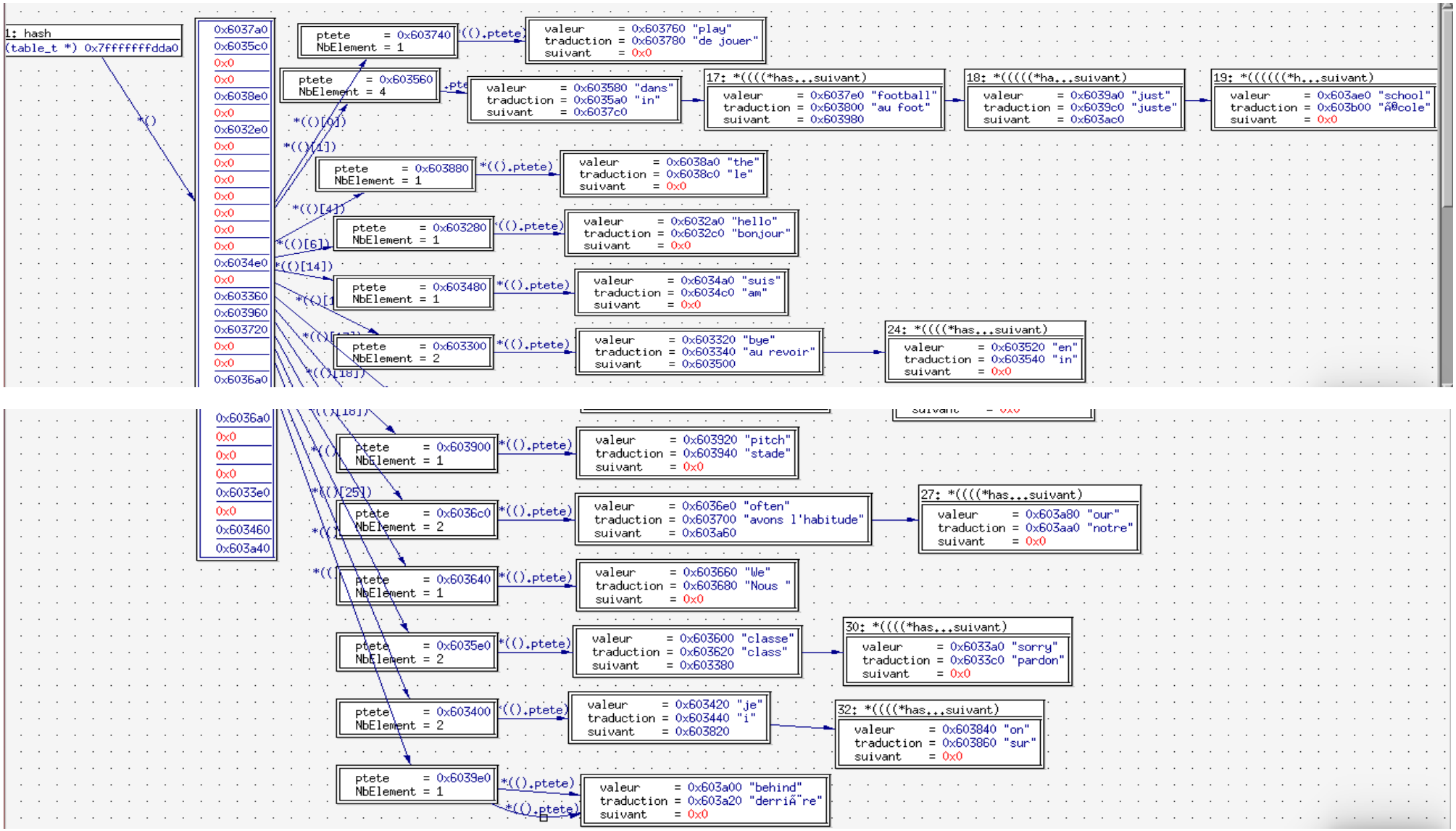
```
cedric@cedric-X302LA:~/TP4_3SDD$ valgrind main dictionnaire_5.txt
==3533== Memcheck, a memory error detector
==3533== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3533== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3533== Command: main dictionnaire_5.txt
==3533==
1. Creation de table.
2. Traduction de la phrase.
2
La trauction de la phrase:
We often play football on the pitch just behind our school
Nous  avons l'habitude de jouer football sur le stade just derrière our school
==3533==
==3533== HEAP SUMMARY:
==3533==     in use at exit: 0 bytes in 0 blocks
==3533==   total heap usage: 59 allocs, 59 frees, 1,330 bytes allocated
==3533==
==3533== All heap blocks were freed -- no leaks are possible
==3533==
==3533== For counts of detected and suppressed errors, rerun with: -v
==3533== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

3.2.2.2 Lorsque tous les mots de la phrase existe dans la table

- **Pour ce cas particulier nous aurons besoins du fichier dictionnaire_4.txt dont son contenu est comme suit :**

```
hello;bonjour
bye;au revoir
sorry;pardon
je;i
suis;am
en;in
dans;in
classe;class
We;Nous
often;avons l'habitude
play;de jouer
football;au foot
on;sur
the;le
pitch;stade
just;juste
behind;derrière
our;notre
school;école
```

- **Résultat de l'exécution avec DDD**



- **Résultat de l'exécution sur le terminal**

```
cedric@cedric-X302LA:~/TP4_3SDD$ main dictionnaire_4.txt
1. Creation de table.
2. Traduction de la phrase.
2
La trauction de la phrase:
We often play football on the pitch just behind our school
Nous  avons l'habitude de jouer au foot sur le stade juste derrière notre école
```

- **Exécution avec valgrind**

```
cedric@cedric-X302LA:~/TP4_3SDD$ valgrind main dictionnaire_4.txt
==3557== Memcheck, a memory error detector
==3557== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3557== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3557== Command: main dictionnaire_4.txt
==3557==
1. Creation de table.
2. Traduction de la phrase.
2
La trauction de la phrase:
We often play football on the pitch just behind our school
Nous  avons l'habitude de jouer au foot sur le stade juste derrière notre école
==3557==
==3557== HEAP SUMMARY:
==3557==     in use at exit: 0 bytes in 0 blocks
==3557==   total heap usage: 71 allocs, 71 frees, 1,478 bytes allocated
==3557==
==3557== All heap blocks were freed -- no leaks are possible
==3557==
==3557== For counts of detected and suppressed errors, rerun with: -v
==3557== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```