

# **COMPTE RENDU DU TP3 DE SDD**

**Binôme :**

**AZEROUAL** Mohammed

**LINGOM NKAMGA** David Cédric

## Sommaire

1	Présentation générale .....	3
1.1	Description de l'objet du TP .....	3
1.2	Description de la structure de données .....	3
1.3	Schéma de la pile .....	3
1.4	Schéma du dictionnaire (liste chaînées à plusieurs niveaux) .....	4
1.5	Description des fichiers des données utilisées (en entrée) .....	5
1.6	Organisation du code source .....	5
2	Détail de chaque fonction .....	10
3	Compte rendu d'exécution .....	49
3.1	Makefile .....	49
3.2	Jeux de test complets .....	50
3.2.1	Création et affichage du dictionnaire .....	50
3.2.2	Création du dictionnaire et affichage des mots qui commencent par un motif .....	63

# 1 Présentation générale

## 1.1 Description de l'objet du TP

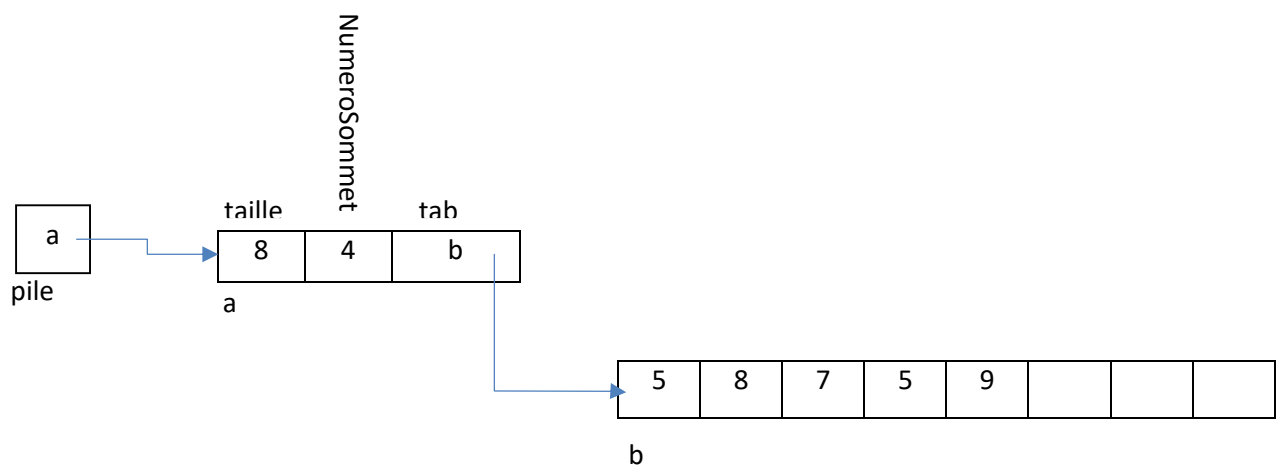
Dans ce TP notre travail consiste à réaliser un dictionnaire qui sera représenté sous la forme d'une structure arborescente qui dans notre cas est une forêt c'est-à-dire un ensemble d'arbres où chaque nœud est une lettre. Pour cela nous devons implémenter des fonctions et procédures qui nous permettront d'insérer les mots lus à partir d'un fichier texte dans le dictionnaire tout en s'assurant qu'il y a unicité des mots dans le dictionnaire. Après nous devons afficher les mots qui ont été insérés dans le dictionnaire et censés être en ordre alphabétique. Mais aussi, nous devons donner les mots du dictionnaire qui commencent par un motif donné.

## 1.2 Description de la structure de données

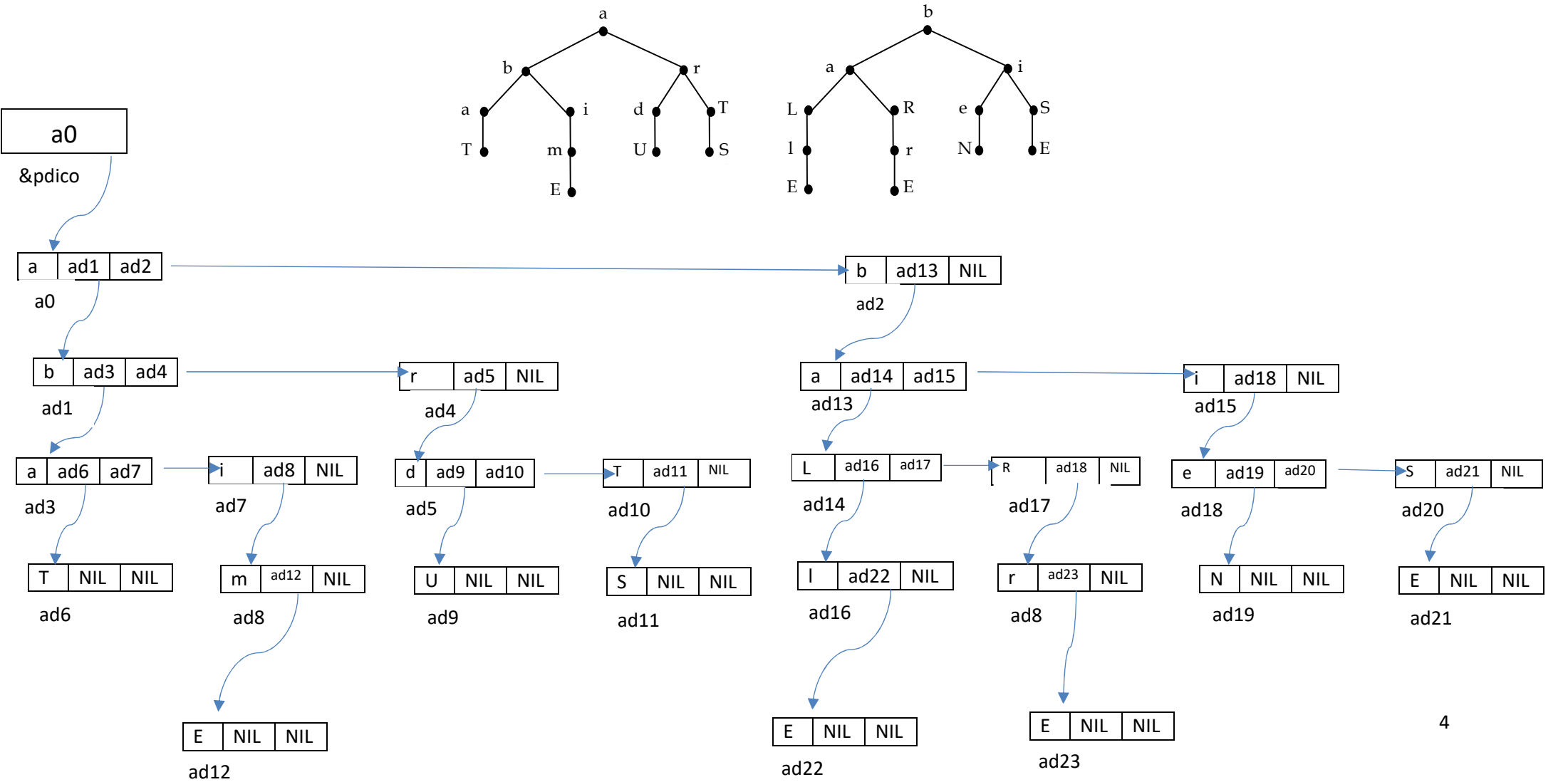
Comme déjà dit plus haut la structure de données utilisée est l'arbre mais pas qu'un seul arbre, parce que le dictionnaire est représenté sous la forme d'une forêt c'est-à-dire un ensemble d'arbres. Cette forêt est en d'autres termes une liste chaînée à plusieurs niveaux et chaque nœud de l'arbre est une lettre alphabétique de type « lettre\_t » qui est une structure à trois champs définie dans le fichier d'entête « lettre.h ». Le premier champ contient la lettre, le deuxième est un lien vertical (pointeur) vers une autre lettre et le troisième un lien horizontal (pointeur) vers une autre lettre. Aussi les listes chaînées par lien horizontal contiennent des lettres qui sont ordonnées par ordre alphabétique.

Aussi comme autre structure de données qui sera utilisée ici est une pile. Sa structure est telle que nous avons un pointeur sur une structure de pile dont le type est « pile\_t » définie dans le fichier « pile.h » et dans cette structure nous avons trois champs. Le premier champ représente la taille de la pile, le deuxième champ « NumeroSommet » nous donne le numéro de l'élément au sommet de la pile, le troisième champ « tab » est un pointeur sur un tableau contenant les éléments de la pile qui sont de type « objet\_t ». Le type « objet\_t » est défini dans le fichier « pile.h ».

## 1.3 Schéma de la pile



1.4 Schéma du dictionnaire (liste chaînées à plusieurs niveaux)



## 1.5 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, dictionnaire\_6.txt, dictionnaire\_7.txt, dictionnaire\_8.txt et cela sera passé en ligne de commande. L'affiche du dictionnaire avec ou sans motif se fera en sortie standard.

## 1.6 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 mots à être insérer dans le dictionnaire à partir d'un fichier texte). Sur ce dictionnaire sera effectuer des traitements grâce aux procédures : « AffichageDico » (affiche les mots dictionnaire), « RechercheDico » (qui recherche un motif donné et affiche les mots qui commencent par ce motif) et « LibererDico » (qui libère le dictionnaire). Notre fichier main.c contient comme fichier d'entête « dico.h » qui est aussi inclus dans le fichier « dico.c » et ce fichier d'entête contient des inclusions, des définitions de constantes symboliques et des prototypes comme suit :

```
/*-----*/
/*          dico.h          */
/*          */
/* Role : Déclaration des directives de compilation, des constantes symboliques, */
/*          des types personnels et des prototypes.          */
/*          */
/*-----*/
```

```
#ifndef _GESTION_DICTIONNAIRE_ARBORESCENTE_DICO_H
```

```
#define _GESTION_DICTIONNAIRE_ARBORESCENTE_DICO_H
```

```
#include <string.h>
```

```
#include "../pile.h"
```

```
#define TAILLEMOT 50
```

```
void LectureMot(FILE *, char *, enum bool *);

lettre_t ** RechercheDivergence(char *, int *, lettre_t **);

void InsertionMot(char *, int *, lettre_t **, enum bool *);

void CreerDico(lettre_t **, FILE *, enum bool *);

void LectureFichier(char *, lettre_t **, enum bool *);

void AffichageDicoAvecMotif(lettre_t *, char *);

void AffichageDico(lettre_t *);

void RechercheMotif(lettre_t *, char *);

void RechercheDico(lettre_t *);

void LibererDico(lettre_t **);
```

```
#endif
```

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

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

/*                                pile.h                                */

/*                                                                */

/* Role : Déclaration des directives de compilation, constantes symboliques, types */

/*      et prototypes.                                          */

/*                                                                */

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

```

#ifndef _GESTION_DICTIONNAIRE_ARBORESCENT_PILE_H

#define _GESTION_DICTIONNAIRE_ARBORESCENT_PILE_H


#include "./lettre.h"


#define TAILLE_MAX 40      /*Taille utiliser pour tester les fonctions de pile*/


typedef lettre_t * objet_t;

typedef struct pile
{
    int taille;

    int NumeroSommet;

    objet_t * tab;
}pile_t;


pile_t * InitialisationPile(int);

void empiler(pile_t *, objet_t);

void EstVide(pile_t *, enum bool *);

objet_t depiler(pile_t *);

objet_t SommetPile(pile_t *);

void AffichageContenuPile(pile_t *);

void LibererPile(pile_t **);

#endif

```

Ensuite dans ce fichier « pile.h » est inclus le fichier « lettre.h ». Le fichier d'entête « lettre.h » est aussi inclus dans « lettre.c » et ce fichier d'entête contient des inclusions tels que « stdio.h » et « stdlib.h », des définitions de constantes symboliques, des définitions de type personnels et des prototypes comme suit :

```
/*-----*/  
/*          lettre.h          */  
/*          */  
/* Role : Déclaration des directives de compilation, constantes symboliques, types */  
/* et prototypes.          */  
/*          */  
/*-----*/
```

```
#ifndef _GESTION_DICTIONNAIRE_ARBORESCENT_LETTRE_H
```

```
#define _GESTION_DICTIONNAIRE_ARBORESCENT_LETTRE_H
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define TAILLELETTRE 3
```



```

typedef struct lettre
{
    char valeur;

    struct lettre * lv;

    struct lettre * lh;
}lettre_t;

enum bool{faux, vrai};


lettre_t * AllocationLettre();

enum bool EstMiniscule(lettre_t *);

enum bool EstMajuscule(lettre_t *);

void EnMajuscule(lettre_t *);

void EnMinuscule(lettre_t *);

void CompareLettre(char, lettre_t *, int *);

lettre_t ** RechercherPrec (lettre_t **, char, enum bool *);

void InsertionLettre (lettre_t **, lettre_t *);

void SuppressionLettre(lettre_t **);


#endif

```

## 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 « lettre.c » sont détaillés comme suit:

```
/*-----*/
/*
/*
/*          lettre.c
/*
/*
/* Role : Définition des procédures et fonctions permettant la gestion des lettres.
/*
/*
/*-----*/
```

```
#include "./lettre.h"
```

```
/*-----*/
/*
/*
/* AllocationLettre      Alloue le bloc devant contenir la lettre.
/*
/*
/* En entrée           : Rien en entrée.
/*
/*
/* En sortie           : pcar - Pointeur du bloc alloué devant contenir la lettre.
/*
/*
/* Variable(s) locale(s) : pcar - Pointeur du bloc alloué devant contenir la lettre
/*
/*
/*-----*/
```

```

lettre_t * AllocationLettre()
{
    lettre_t * pcar = (lettre_t *)malloc(TAILLELETTRE * sizeof(lettre_t));

    if (pcar != NULL)                                /*si l'allocation c'est bien passé*/
    {
        pcar->lv = NULL;
        pcar->lh = NULL;
    }

    return pcar;
}

/*-----*/
/*                                                    */
/* EstMiniscule      Test si le lettre en minuscule.    */
/*                                                    */
/* En entrée        : pcar    - Pointeur sur le bloc contenant la lettre à tester. */
/*                                                    */
/* En sortie        : minuscule - Contentant vrai si la lettre est en minuscule et */
/*                  faux si la lettre est en majuscule. */
/*                                                    */
/* Variable(s) locale(s) : minuscule - Contentant vrai si la lettre est en minuscule et */
/*                  faux si la lettre est en majuscule. */
/*                                                    */
/*-----*/

```

```

enum bool EstMiniscule(lettre_t * pcar)
{
    enum bool minuscule = faux;          /*on suppose qu'il n'est pas minuscule*/
    if((pcar->valeur >= 'a') && (pcar->valeur <= 'z'))    /*s'il est minuscule*/
    {
        minuscule = vrai;
    }
    return minuscule;
}

```

```

/*-----*/
/*                                                    */
/* EstMajuscule      Test si la lettre en majuscule.    */
/*                                                    */
/* En entrée        : pcar    - Pointeur sur le bloc contenant la lettre à tester. */
/*                                                    */
/* En sortie        : majuscule - Contentant vrai si la lettre est en majuscule et */
/*                  faux si la lettre est en minuscule. */
/*                                                    */
/* Variable(s) locale(s) : majuscule - Contentant vrai si la lettre est en majuscule et */
/*                  faux si la lettre est en minuscule. */
/*                                                    */
/*-----*/

```

```

enum bool EstMajuscule(lettre_t * pcar)
{
    enum bool majuscule = faux;          /*on suppose qu'il n'est pas en majuscule*/

    if((pcar->valeur >= 'A') && (pcar->valeur <= 'Z')) /*s'il est minuscule*/
    {
        majuscule = vrai;
    }

    return majuscule;
}

```

```

/*-----*/
/*
/* EnMajuscule      Transforme une lettre en majuscule.
/*
/*
/* En entrée       : pcar - Pointeur sur le bloc contenant la lettre.
/*
/*
/* En sortie       : pcar - Pointeur sur le bloc contenant la lettre.
/*
/*
/* Variable(s) locale(s) : Rien en variable locale.
/*
/*
/*-----*/

```

```
void EnMajuscule(lettre_t * pcar)
```

```
{
```

```
    pcar->valeur = pcar->valeur - ('a' - 'A');
```

```
}
```

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

```
/*                                                    */
```

```
/* EnMinuscule      Transforme une lettre en minuscule. */
```

```
/*                                                    */
```

```
/* En entrée       : pcar - Pointeur sur le bloc contenant la lettre. */
```

```
/*                                                    */
```

```
/* En sortie       : pcar - Pointeur sur le bloc contenant la lettre. */
```

```
/*                                                    */
```

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

```
/*                                                    */
```

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

```
void EnMinuscule(lettre_t * pcar)
```

```
{
```

```
    pcar->valeur = pcar->valeur + ('a' - 'A');
```

```
}
```

```

/*-----*/
/*
*/
/* CompareLettre      Compare deux lettres sans tenir compte des majuscules. */
/*
*/
/* En entrée      : c      - Variable contenant la première lettre. */
/*
*/      pcar      - Pointeur sur un bloc contenant la deuxième lettre. */
/*
*/      comparaison - Pointeur sur une case mémoire contenant une */
/*
*/      valeur positive si la première lettre > la deuxième, */
/*
*/      une valeur négative si la première lettre < la */
/*
*/      deuxième et zéro s'ils sont égaux. */
/*
*/
/* En sortie      : comparaison - Pointeur sur une case mémoire contenant une */
/*
*/      valeur positive si la première lettre > */
/*
*/      la deuxième, une valeur négative si la première */
/*
*/      lettre < la deuxième et zéro s'ils sont égaux. */
/*
*/
/* Variable(s) locale(s) :      Rien comme variable locale. */
/*
*/
/*-----*/

```

```

void CompareLettre(char c, lettre_t * pcar, int * comparaison)
{
    *comparaison = -1;

    if(pcar)
    {
        if(EstMajuscule(pcar))
        {
            *comparaison = c - (pcar->valeur + ('a' - 'A'));
        }
        else
        {
            *comparaison = c - pcar->valeur;
        }
    }
}

```

```

/*-----*/
/*
/* RecherchePrec      Recherche une lettre dans la liste chaînée des lettres.
/*
/*
/* En entrée      : PpteteListe - Pointeur de pointeur de tête de la liste chaînée des
/*                  lettres.
/*
/*      lettre      - Pointeur sur la lettre à rechercher.
/*
/*      ptrouver     - Pointeur sur une case mémoire contenant 0 si on a
/*                  pas trouvé ou 1 si on a trouvé la lettre.
/*
/*

```



```

/* En sortie      : prec      - Retourne l'adresse du pointeur de tete de liste      */
/*
/*              chaînée des lettres ou l'adresse de la case pointeur de */
/*              l'élément précédent dans la liste chaînée des lettres.  */
/*
/*      ptrouver - Pointeur sur une case mémoire contenant 0 si on a pas*/
/*
/*              trouvé l'élément ou 1 si on a trouvé dans la liste      */
/*
/*              chaînée des lettres.                                     */
/*
/*
/*
/* Variable(s) locale(s) : pcour      - Pointeur sur la lettre courante.      */
/*
/*      prec      - Pointeur de pointeur de tete de liste chaînée      */
/*
/*              des lettres ou pointeur sur la case pointeur de      */
/*
/*              l'élément précédent de la liste chaînée des      */
/*
/*              lettres.                                               */
/*
/*      comparaison - Variable le résultat de la comparaison de deux      */
/*
/*              lettres qui sera positif si la première lettre >      */
/*
/*              la deuxième lettre, négatif si la première lettre<*/
/*
/*              la deuxième lettre et zéro s'ils sont égaux.          */
/*
/*
/*
/*-----*/

```

```

lettre_t ** RechercherPrec (lettre_t ** PpteteListe, char lettre, enum bool * ptrouver)
{
    int comparaison;

    lettre_t * pcour = *PpteteListe, ** prec = PpteteListe;          /*Initialisation à la première
lettre et au pointeur de tete*/

    *ptrouver = faux;

    CompareLettre(lettre, pcour, &comparaison);

    while((pcour != NULL) && (comparaison > 0)) /*Tantque je suis dans la liste et que ma lettre
est plus grande*/
    {
        prec = &(pcour->lh);          /*On récupère l'adresse de la case pointeur
de l'élément courant*/

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

        CompareLettre(lettre, pcour, &comparaison);
    }

    if ((pcour != NULL) && !(comparaison))    /*Si on trouve la lettre*/
    {
        *ptrouver = vrai;
    }

    return prec;
}

```

```

/*-----*/
/*
*/
/* InsertionLettre      Insère une nouvelle lettre dans la liste chaînée des lettres. */
/*
*/
/* En entrée          : ppcar - Pointeur de pointeur de tete de liste chaînée des lettres ou */
/*
/*                    pointeur sur la case pointeur de l'élément précédent. */
/*
/*                    pcar - Pointeur sur le bloc contenant la lettre à insérer. */
/*
*/
/* En sortie          : ppcar - Pointeur de pointeur de tete de liste chaînée des lettres ou */
/*
/*                    pointeur sur la case pointeur de l'élément précédent. */
/*
*/
/* Variable(s) locale(s) : Rien en variable locale. */
/*
*/
/*-----*/

```

```

void InsertionLettre (lettre_t ** ppcar, lettre_t * pcar)
{
    pcar->lh = *ppcar;
    *ppcar = pcar;
    pcar = NULL;
}

/*-----*/
/*                                                    */
/* SuppressionLettre    Supprime une lettre de liste chaînée courante des lettres. */
/*                                                    */
/* En entrée          : ppcar - Pointeur de pointeur de tête de liste chaînée courante des */
/*                      lettres ou pointeur sur la case pointeur de l'élément */
/*                      précédent de la liste chaînée courante des lettres. */
/*                      */
/* En sortie          : ppcar - Pointeur de pointeur de tête de liste chaînée courante des */
/*                      lettres ou pointeur sur la case pointeur de l'élément */
/*                      précédent de la liste chaînée courante des lettres. */
/* Variable(s) locale(s) : pcar - Pointeur sur lettre à supprimer. */
/*                      */
/*-----*/

```

```

void SuppressionLettre(lettre_t ** ppcar)
{
    lettre_t * pcar = *ppcar; /*recupère l'adresse de lettre à supprimer*/
    *ppcar = pcar->lh;        /*pointe sur la lettre après la lettre à supprimer*/
    free(pcar);               /*supprime la lettre*/
    pcar = NULL;
}

```

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

```

/*-----*/
/*                                pile.c                                */
/*                                                                */
/* Role : Définitions des fonctions et procédures permettant la gestion des piles. */
/*                                                                */
/*-----*/

```

```

#include "./pile.h"

```

```

/*-----*/
/* InitialisationPile    Alloue une structure pile ainsi que le tableau servant de pile. */
/*
/*
/* En entrée :    TaillePile - Elle représente la taille de la taille.
/*
/*
/* En sortie :    Renvoie l'adresse de la structure de pile.
/*
/*
/* Variable(s) locale(s) :    pile - Pointeur sur la structure de pile.
/*
/*
/*-----*/

```

```

pile_t * InitialisationPile (int TaillePile)

```

```

{
    pile_t * pile = (pile_t *)malloc(sizeof(pile_t));

    if(pile != NULL)                                /* si l'allocation est reussite*/
    {
        pile->taille = TaillePile;                    /*fixer la taille de la pile*/
        pile->NumeroSommet = -1;
        pile->tab = (objet_t *)malloc(TaillePile * sizeof(objet_t));    /* allouer un tableau
de donnees de taille TaillePile */
        if(pile->tab == NULL)                        /* si l'allocation de tab est echouée*/
        {
            free(pile);                                /* on libere la Pile */
            pile = NULL;
        }
    }

    return pile;
}

```

```
}
```

```
/*-----*/  
/* empiler    Empile une valeur dans la pile (Ajoute une valeur au sommet de la pile).*/  
/*                                                    */  
/* En entrée : pile - Pointeur sur la structure de pile. */  
/*            val - La valeur à empiler.                */  
/*                                                    */  
/* En sortie : Rien en sortie.                        */  
/*                                                    */  
/*-----*/
```

```
void empiler(pile_t * pile, objet_t val)
```

```
{  
    (pile->tab)[(pile->NumeroSommet) + 1] = val;    /*ajouter la valeur à la pile */  
    ++pile->NumeroSommet;                          /*augmenter le nombre d'element de la pile */  
}
```

```

/*-----*/
/* EstVide    Vérifie si la pile vide.                */
/*                                                    */
/* En entrée :  pile - Pointeur sur la structure de pile. */
/*            vide - Pointeur sur la case mémoire contenant comme valeur vrai ou */
/*            faux, pour dire comme quoi la pile est respectivement vide ou pas.*/
/*                                                    */
/* En sortie :  vide - Pointeur sur la case mémoire contenant comme valeur vrai ou */
/*            faux, pour dire comme quoi la pile est respectivement vide ou pas. */
/*                                                    */
/*-----*/

```

```

void EstVide(pile_t * pile , enum bool * vide)

```

```

{
    *vide = faux;

    if(pile->NumeroSommet == -1)
    {
        *vide = vrai ;
    }
}

```



```

/*-----*/
/* depiler    Retire la valeur au sommet et la supprime de la pile.          */
/*                                                    */
/* En entrée : pile - Pointeur de la structure de pile.                      */
/*                                                    */
/* En sortie : Retourne le sommet de la pile.                                */
/*                                                    */
/* Variable(s) locale(s) :  sommet - Contenant la valeur du sommet de la pile à être */
/*                               supprimer de la pile.                          */
/*                                                    */
/*-----*/

```

```

objet_t depiler(pile_t * pile)

```

```

{
    objet_t sommet = (pile->tab)[pile->NumeroSommet]; /* sauvegarder le sommet*/
    - pile->NumeroSommet;                               /*deminuer le nombre d'element de la pile */

    return sommet;
}

```

```

/*-----*/
/* SommetPile      Renvoie le sommet de la pile.          */
/*                                                         */
/*                                                         */
/* En entrée :   pile - Pointeur sur la structure de pile. */
/*                                                         */
/*                                                         */
/* En sortie :   La fonction retourne le sommet de la pile.*/
/*                                                         */
/*                                                         */
/*-----*/

```

```

objet_t SommetPile(pile_t * pile)
{
    return (pile->tab)[pile->NumeroSommet];
}

```

```

/*-----*/
/* AfficherContenuPile    Affiche le contenu des pointeurs dans de notre pi */
/*                                                         */
/*                                                         */
/* En entrée      : pile - Pointeur sur la structure de pile. */
/*                                                         */
/*                                                         */
/* En sortie      :      Rien en sortie. */
/*                                                         */
/*                                                         */
/* Variable(s) locale(s) : i - Variable servant de variable de boucle. */
/*                                                         */
/*                                                         */
/*-----*/

```

```

void AffichageContenuPile(pile_t * pile)
{
    int i;

    if(pmotif) /*si le motif existe*/
    {
        printf("%s", pmotif);
    }*/
    for(i = 0; i <= pile->NumeroSommet; ++i)
    {
        if(EstMajuscule((pile->tab)[i])) /*si c'est une majuscule*/
        {
            printf("%c",(((pile->tab)[i])->valeur) + ('a' - 'A'));
        }
        else
        {
            printf("%c",((pile->tab)[i])->valeur);
        }
    }
}

```

```

/*-----*/
/* LibererPile    Libère la pile.                                */
/*                                                       */
/* En entrée :   ppile - Pointeur de pointeur sur la structure de pile. */
/*                                                       */
/* En sortie :   ppile - L'adresse du pointeur sur la structure de pile. */
/*                                                       */
/* Variable(s) locale(s) : Rien en variable locale          */
/*                                                       */
/*-----*/

```

```

void LibererPile(pile_t ** ppile)
{
    free((*ppile)->tab);    /* désallouer le tab de donnees(implicitement la pile)*/
    free(*ppile);           /* désallouer la structure de pile */
    *ppile = NULL;
}

```

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

```

/*-----*/
/*                                                    */
/*                                                    */
/*                                                    */
/*                                                    */
/* Role : Définition des procédures et fonctions permettant la gestion du dictionnaire */
/*                                                    */
/*-----*/

```

```

#include "./dico.h"

```

```

/*-----*/
/*                                                    */
/*                                                    */
/* LectureMot      Lit un mot à partir d'un fichier.      */
/*                                                    */
/*                                                    */
/* En entrée      : f          - Pointeur sur un fichier.  */
/*                                                    */
/*                pmot        - Pointeur sur une chaîne de caractères qui est un */
/*                mot.          */
/*                                                    */
/*                PcodeLecture - Pointeur sur une case mémoire contenant une */
/*                valeur 1 si la lecture c'est bien passée et 0 sinon. */
/*                                                    */
/* En sortie      : pmot        - Pointeur sur une chaîne de caractères qui est un */
/*                mot.          */
/*                                                    */
/*                PcodeLecture - Pointeur sur une case mémoire contenant une */
/*                valeur 1 si la lecture c'est bien passée et 0 sinon. */
/*                                                    */
/*                                                    */

```

```

/* Variable(s) locale(s) :          Rien comme variable locale.          */
/*                                                                           */
/*                                                                           */
/*-----*/

void LectureMot(FILE * f, char * pmot, enum bool * PcodeLecture)
{
    if(fgets(pmot, TAILLEMOT, f)) /*si un mot existe dans le fichier et que la lecture c'est bien
passé*/
    {
        *PcodeLecture = vrai;
    }
    else
    {
        *PcodeLecture = faux;
    }
}

/*-----*/
/*                                                                           */
/* RechercheDivergence      Recherche et renvoie l'adresse de la case pointeur de */
/*                             la lettre après la quelle on doit insérer.          */
/*                                                                           */
/* En entrée      : pmot      - Pointeur sur le mot à insérer dans le dictionnaire. */
/*                  position   - L'indice de la lettre courante dans le mot.      */
/*                  PpteteListe - Pointeur de pointeur de liste chaînées de lettres. */
/*                                                                           */

```

```

/* En sortie :      position      - L'indice de la lettre courante dans le mot.      */
/*
/*      PpteteListe - Pointeur de pointeur de liste chaînées de lettres.      */
/*
/*
/* Variable(s) locale(s) : prec      - Pointeur de pointeur de tête de liste chaînée de */
/*
/*      lettres ou pointeur sur la case pointeur de      */
/*
/*      l'élément précédent de la liste chaînée de      */
/*
/*      lettres.      */
/*
/*      trouver      - Case mémoire contenant vrai si la lettre est      */
/*
/*      trouver et faux sinon.      */
/*
/*
/*-----*/

```

```

lettre_t ** RechercheDivergence(char * pmot, int * position, lettre_t ** PpteteListe)
{
    enum bool trouver = vrai;

    lettre_t ** prec = PpteteListe;

    while(((pmot[*position] != '\n') && (pmot[*position] != '\0')) && (trouver))/*tantqu'on est
pas à la fin du mot et qu'on trouve que la lettre précédent du mot existe déjà */
    {
        prec = RechercherPrec(prec, pmot[*position], &trouver);      /* Recherche la lettre
courante du mot dans la liste chaînée courante*/

        if(trouver)      /*si on a trouvé la lettre courante*/
        {
            (*position)++;      /*on avance dans le mot*/

            if((pmot[*position] != '\n') && (pmot[*position] != '\0')) /*si on est pas à la
fin du mot*/
            {
                prec = &((*prec)->lv);      /*On va vers le lien vertical*/
            }
        }
    }
}

```

```

        }

    }

}

return prec;

}

/*-----*/

/*                                                    */

/* InsertionMot      Insère un mot dans le dictionnaire. */

/*                                                    */

/* En entrée      : pmot      - Pointeur sur le mot à insérer dans le dictionnaire. */
/*                position    - Pointeur sur la case mémoire contenant la          */
/*                               position courante dans le mot.                    */
/*                ppcar       - Pointeur de pointeur de tête de liste chaînée des */
/*                               lettres ou pointeur sur la case pointeur de          */
/*                               l'élément précédent de liste chaînée de lettres. */
/*                PcodeInsertion - Pointeur sur la case mémoire contenant vrai si */
/*                               l'insertion c'est bien passée et faux sinon.      */
/*                                                    */

/* En sortie      : position    - Pointeur sur la case mémoire contenant la          */
/*                               position courante dans le mot.                    */
/*                PcodeInsertion - Pointeur sur la case mémoire contenant vrai si */
/*                               l'insertion c'est bien passée et faux sinon.      */
/*                                                    */

/* Variable(s) locale(s) : pcar      - Pointeur sur le bloc devant contenir une lettre du */
/*                                     mot.                                          */
/*                                                    */

/*-----*/

```





```

void InsertionMot(char * pmot, int * position, lettre_t ** ppcar, enum bool * PcodeInsertion)
{
    lettre_t * pcar = NULL;

    *PcodeInsertion = vrai;          /*on suppose que l'insertion du mot va bien se passé*/

    while(pmot[*position] != '\n')    /*si on est pas à la fin du mot à insérer*/
    {
        pcar = AllocationLettre();

        if(pcar != NULL)              /*si le bloc lettre est alloué*/
        {
            pcar->valeur = pmot[*position];    /*insérer la lettre dans le bloc*/
            InsertionLettre(ppcar, pcar);        /*insère le bloc dans la liste chaînée
courante de lettres*/
            (*position)++;                      /*on avance dans le mot*/
            if(pmot[*position] != '\n')          /*si on n'est pas à la fin du mot*/
            {
                ppcar = &((*ppcar)->lv);        /*on fait la mise à jour du pointeur
precedent*/
            }
        }
        else
        {
            *PcodeInsertion = faux;
        }
    }

    if(EstMiniscule(*ppcar))
    {
        EnMajuscule(*ppcar);
    }
}

```

}

```
/*-----*/  
/* */  
/* CreerDico      Création du dictionnaire. */  
/* */  
/* En entrée      : PpteteListe  - Pointeur de pointeur de tête de liste chaînée de */  
/*                lettres. */  
/*                f              - Pointeur sur un fichier. */  
/*                PcodeCreation - Pointeur sur la case mémoire contenant vrai si la */  
/*                création du dictionnaire c'est bien passé et faux */  
/*                sinon. */  
/* */  
/* En sortie      : PpteteListe  - Pointeur de pointeur de tête de liste chaînée de */  
/*                lettres. */  
/*                PcodeCreation - Pointeur sur la case mémoire contenant vrai si la */  
/*                création du dictionnaire c'est bien passé et faux */  
/*                sinon. */  
/* */  
/* Variable(s) locale(s) : position - Position courante dans le mot c'est-à-dire l'indice */  
/*                CodeLecture  - Case mémoire contenant faux si on a atteint */  
/*                la fin du fichier et vrai sinon. */  
/*                CodeInsertion - Case mémoire contenant vrai si l'insertion */  
/*                d'un mot c'est bien passé et faux sinon. */  
/*                pmot         - Pointeur sur le mot (chaîne de caractères). */  
/*                prec         - Pointeur de pointeur de tête de liste chaînée */
```

```

/*                                de lettre ou pointeur de la case pointeur de */
/*                                l'élément précédent de la liste chaînée des */
/*                                lettres.                                */
/*                                */
/*-----*/

```

```

void CreerDico(lettre_t ** PpteteListe, FILE * f, enum bool * PcodeCreation)
{
    int position = 0;

    enum bool CodeLecture = vrai, CodeInsertion = vrai;

    char * pmot = (char *)malloc(TAILLEMOT * sizeof(char));

    lettre_t ** prec = NULL;

    *PcodeCreation = vrai;                                /*on suppose que la création va bien se passé*/

    if(pmot != NULL)                                    /*si l'allocation c'est bien passé*/
    {
        LectureMot(f, pmot, &CodeLecture);

        while(CodeLecture && CodeInsertion)                /*tantque la lecture continue et
l'insertion se passe bien*/
        {
            position = 0;

            prec = RechercheDivergence(pmot, &position, PpteteListe);

            InsertionMot(pmot, &position, prec, &CodeInsertion);

            if(CodeInsertion)                                /*si l'insertion se passe bien*/
            {
                LectureMot(f, pmot, &CodeLecture);
            }
        }
    }
}

```

```

        if(!CodeInsertion)
        {
            printf("Erreur d'insertion d'un mot dans le dictionnaire.\n");
            *PcodeCreation = faux;
        }
        free(pmot);
    }
else
{
    printf("Erreur dans la création du dictionnaire!\nBloc lettre non alloué!\n");
    *PcodeCreation = faux;
}
}

/*-----*/
/*
/*
/* LectureFichier    Permet l'ouverture du fichier et la création du dictionnaire des mots lu à
/*
/*                  partir du fichier.
/*
/*
/* En entrée: NomFichier - Nom du fichier dans le quel on va lire.
/*
/*      PpteteListe - Pointeur de pointeur de tête de liste chaînée des semaines.
/*
/*      PcodeLecture - Pointeur sur la case contenant le code de lecture vrai si la lecture
/*
/*                  c'est bien passé et faux sinon.
/*
/*
/* En sortie: PpteteListe - Pointeur de pointeur de tête de liste chaînée des semaines.
/*
/*      PcodeLecture - Pointeur sur la case contenant le code de lecture vrai si la lecture

```

```

/*          c'est bien passé et faux sinon.          */
/*          */
/* Variable(s) locale(s): f - Pointeur sur le fichier. */
/*          */
/*-----*/

```

```

void LectureFichier(char * NomFichier, lettre_t ** PpteteListe, enum bool * PcodeLecture)
{
    FILE * f = NULL;

    *PcodeLecture = vrai;          /*on suppose que la lecture va bien se passée*/

    f = fopen(NomFichier,"r");

    if (f != NULL)                /*si le fichier est ouvert*/
    {
        CreerDico(PpteteListe, f, PcodeLecture);    /*Crée un dictionnaire*/
        fclose(f);
    }
    else                          /*Si la lecture ne s'est pas bien passé*/
    {
        *PcodeLecture = faux;
        printf("Erreur d'ouverture du fichier!\n");
    }
}

```

```

/*-----*/
/*
/* AffichageDicoAvecMotif      Affichage de tout les mots de dictionnaire en
/*                               ordre alphabétique.
/*
/*
/* En entrée      : pdico      - Pointeur de tête de liste chaînée de lettres.
/*                  pmotif     - Pointeur sur une chaine de caractères.
/*
/*
/* En sortie      :            Rien en sortie
/*
/*
/* Variable(s) locale(s) : pcour - L'élément courant qui parcourt l'arbre.
/*                               pile - Pointeur sur la structure de pile contient les
/*                               adresses des lettres parcourues.
/*
/*                               fin - Un booléen indique la fin du programme
/*
/*
/*-----*/

```

```

void AffichageDicoAvecMotif(lettre_t * pdico, char * pmotif)
{
    lettre_t * pcour = pdico;

    enum bool fin = faux;                /*initialialisation de notre courant*/

    pile_t * pile = InitialisationPile(TAILLE_MAX); /* l'allocation de la pile*/

    if(pile)                             /* si l'allocation est bien fait*/
    {
        while(!fin)                      /* tant que on est pas arrivé à la fin*/
        {

```

```

while(pcour != NULL)          /*tant que le courant est different de NIL*/
{
    empiler(pile, pcour);    /* on empile l'adresse de la lettre courante*/
    if(EstMajuscule(pcour))    /*si la lettre est en majuscule*/
    {
        if(pmotif)
        {
            printf("%s", pmotif);
        }
        AffichageContenuPile(pile);    /* On affiche tous lettre dont
les pointeurs sont dans la pile*/
        printf("\n");
    }
    pcour = pcour->lv;        /* on avance vers le lien vertical*/
}
EstVide(pile,&fin);
if(!fin)                    /*si la pile n'est pas vide*/
{
    pcour = depiler(pile);
    pcour = pcour->lh;
}
}
LibererPile(&pile);
}
else
{
    printf("Erreur dans l'allocation de la pile");
}
}

```



```
}
```

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

```
/* */
```

```
/* AffichageDico      Affichage de tout les mots de dictionnaire en ordre */
```

```
/*                  alphabétique. */
```

```
/* */
```

```
/* En entrée      : pdico      - Pointeur de tête de liste chaînée de lettres. */
```

```
/* */
```

```
/* En sortie      :              Rien en sortie */
```

```
/* */
```

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

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

```
void AffichageDico(lettre_t * pdico)
```

```
{
```

```
    printf("Le contenu du dictionnaire est :\n");
```

```
    AffichageDicoAvecMotif(pdico, NULL);
```

```
}
```

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

```
/* LibererDico      Libère le dictionnaire (qui est une forêt). */
```

```
/* */
```

```
/* En entrée      : PpteteListe - Pointeur de pointeur de tête de liste chaînée des lettres. */
```

```
/* */
```

```

/* En sortie      : PpteteListe - Pointeur de pointeur de tête de liste chaînée des lettres.    */
/*                                                         */
/* Variable(s) locale(s) : vide      - Pointeur sur une case mémoire contenant vrai si la pile vide */
/*                                                         */
/*                                                         et faux sinon.                      */
/*                                                         */
/*      pile      - Pointeur de structure de pile                      */
/*                                                         */
/*      pcour     - Pointeur de lettre courante dans la liste chaînée de lettres. */
/*                                                         */
/*                                                         */
/*-----*/

```

```

void LibererDico(lettre_t ** PpteteListe)
{
    enum bool vide;

    pile_t * pile = InitialisationPile(TAILLE_MAX);

    lettre_t * pcour = *PpteteListe;

    if(pile)                /*si la pile est alloué*/
    {
        while(pcour != NULL)                /*tantque j'ai pas parcouru tout l'arbre*/
        {
            while(pcour->lv != NULL)        /*tantqu'il a un lien vertical*/
            {
                empiler(pile, pcour); /*sauvegarder l'adresse de la lettre courante*/
                pcour = pcour->lv;      /*Mise à jour du courant*/
            }

            SuppressionLettre(&pcour);

            EstVide(pile, &vide);

            while( !pcour && !vide)        /*tantqu'on a aucune lettre courante et
la pile est non vide*/

```

```

        {
            pcour = depiler(pile);
            SuppressionLettre(&pcour);
            EstVide(pile, &vide);
        }
    }
    LibererPile(&pile);
}

```

```

/*-----*/
/*
/* RechercheMotif      Recherche un motif et affiche les mots qui commencent */
/*                    par ce motif.                                         */
/*
/*
/* En entrée      : pdico      - Pointeur de tête de liste chaînée de lettres. */
/*                pmotif      - Pointeur sur une chaîne de caractères.         */
/*
/* En sortie      :              Rien en sortie                            */
/*
/* Variable(s) locale(s) : position    - La position courante dans le motif.   */
/*                prec      - Pointeur de pointeur sur un bloc contenant une */
/*                lettre.                                         */
/*
/*
/*-----*/

```

```

void RechercheMotif(lettre_t * pdico, char * pmotif)
{
    int position = 0;

    lettre_t ** prec = RechercheDivergence(pmotif, &position, &pdico);    /*Cherche si le motif
existe*/

    if(pmotif[position] == '\0')                                          /*si le motif existe*/
    {
        printf("Le mots commençant par le motif sont:\n");

        if(EstMajuscule(*prec))
        {
            printf("%s\n", pmotif);
        }

        prec = &((*prec)->lv);

        AffichageDicoAvecMotif(*prec, pmotif);    /*Affiche les mots qui commencent par le
motif*/
    }
    else
    {
        printf("Le motif n'existe pas!\n");
    }
}

```

```

/*-----*/

/*                                                                    */
/* RechercheDico      Lit un motif qui sera recherché dans le dictionnaire.    */
/*                                                                    */
/*                                                                    */
/* En entrée      : pdico      - Pointeur de tête de liste chaînée de lettres.    */

```

```

/*                                                                    */
/* En sortie      :          Rien en sortie                          */
/*                                                                    */
/* Variable(s) locale(s) : CodeLecture - Variable contenant vrai si la lecture du motif */
/*                                                                    */
/*                                                                    c'est bien passé et faux sinon.          */
/*                                                                    */
/*          pmotif      - Pointeur sur une chaine de caractères.      */
/*                                                                    */
/*                                                                    */
/*-----*/

```

```

void RechercheDico(lettre_t * pdico)
{
    enum bool CodeLecture;

    char * pmotif = (char *)malloc(TAILLEMOT * sizeof(char));

    if(pmotif)                                /*si l'allocation du motif réussi*/
    {
        printf("Veuillez entrer le motif à rechercher\n");

        CodeLecture = scanf("%s", pmotif);

        if(CodeLecture)                        /*si lecture du motif bien passé*/
        {
            RechercheMotif(pdico, pmotif);
        }

        free(pmotif);
    }

    else

    {

        printf("Erreur d'allocation du motif.\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 à d'autres fonctions et */
/*        procédurés tels que: LectureFichier, AffichageDico, RechercheDico, */
/*        LibérerDico qui permettent des traitements sur le dictionnaire. */
/*                                                    */
/*-----*/

```

```

#include "./dico.h"

```

```

/*-----*/
/*                                                    */
/* main      Fonction principale faisant à des fonctions et procédures qui */
/*            permettent des traitements sur le dictionnaire. */
/*                                                    */
/* En entrée   : argc   - Compte de le nombre d'arguments passés en ligne */
/*              de commande. */
/*            argv  - Tableau de pointeur sur des 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. */
/*                  pdico      - Pointeur de tete de la première liste      */
/*                  chaînée de lettres                                     */
/*                  (liste chaînée des racines).                         */
/*                                                    */
/*-----*/

```

```

int main(int argc, char ** argv)
{
    enum bool CodeLecture;

    lettre_t * pdico = NULL;

    if(argc == 2)
    {
        LectureFichier(argv[1], &pdico, &CodeLecture);

        if(CodeLecture)
        {
            AffichageDico(pdico);      /*Affiche le dictionnaire sans motif*/

            RechercheDico(pdico);      /*Reherche un motif et affiche les mots d'un
dictionnaire qui commence par ce motif*/

            LibererDico(&pdico);

```

```
    }  
    else  
    {  
        printf("Erreur de lecture dans le fichier!");  
    }  
}  
else  
{  
    printf("Veuillez 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
LDFLAGS = -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 et affichage du dictionnaire

#### 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

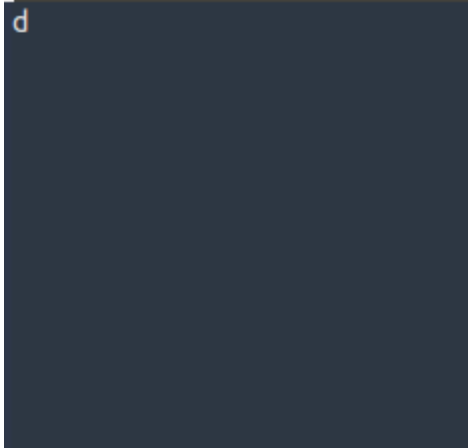
```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_1.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
cedric@cedric-X302LA:~/TP3SDD$
```

- ✚ Exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_1.txt
==2592== Memcheck, a memory error detector
==2592== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2592== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2592== Command: ./main dictionnaire_1.txt
==2592==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
==2592==
==2592== HEAP SUMMARY:
==2592==    in use at exit: 0 bytes in 0 blocks
==2592==   total heap usage: 6 allocs, 6 frees, 1,290 bytes allocated
==2592==
==2592== All heap blocks were freed -- no leaks are possible
==2592==
==2592== For counts of detected and suppressed errors, rerun with: -v
==2592== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cedric@cedric-X302LA:~/TP3SDD$
```

### 3.2.1.2 Lorsque le fichier ne contient qu'une lettre

- ✚ Pour cas la donnée entrée est dans le fichier dictionnaire\_2.txt représenter par cette capture d'écran:



- ✚ Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_2.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
d
cedric@cedric-X302LA:~/TP3SDD$
```

- ✚ Exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_2.txt
==2661== Memcheck, a memory error detector
==2661== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2661== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2661== Command: ./main dictionnaire_2.txt
==2661==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
d
==2661==
==2661== HEAP SUMMARY:
==2661==   in use at exit: 0 bytes in 0 blocks
==2661== total heap usage: 7 allocs, 7 frees, 1,362 bytes allocated
==2661==
==2661== All heap blocks were freed -- no leaks are possible
==2661==
==2661== For counts of detected and suppressed errors, rerun with: -v
==2661== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

### 3.2.1.3 Lorsque le fichier ne contient que des lettres avec certaines qui sont répétées

✚ Les données d'entrées sont dans le fichier dictionnaire\_3.txt comme suit:

```
d
e
a
y
f
b
r
a
g
d
s
```

✚ Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_3.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
a
b
d
e
f
g
r
s
y
cedric@cedric-X302LA:~/TP3SDD$
```

### 🚩 L'exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_3.txt
==2725== Memcheck, a memory error detector
==2725== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2725== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2725== Command: ./main dictionnaire_3.txt
==2725==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
a
b
d
e
f
g
r
s
y
==2725==
==2725== HEAP SUMMARY:
==2725==    in use at exit: 0 bytes in 0 blocks
==2725==   total heap usage: 15 allocs, 15 frees, 1,938 bytes allocated
==2725==
==2725== All heap blocks were freed -- no leaks are possible
==2725==
==2725== For counts of detected and suppressed errors, rerun with: -v
==2725== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

#### 3.2.1.4 Lorsque fichier ne contient que des lettre toutes distinctes

🚩 Les données d'entrées sont dans le fichiers dictionnaire\_4.txt

```
d
e
a
y
f
b
r
g
s
```



## 🚩 Résultat de l'exécution

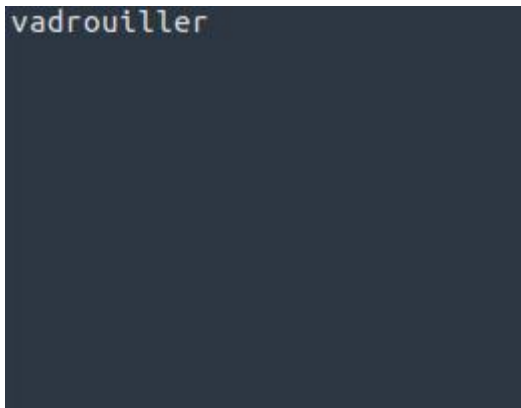
```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_4.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
a
b
d
e
f
g
r
s
y
```

## 🚩 L'exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_4.txt
==2793== Memcheck, a memory error detector
==2793== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2793== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2793== Command: ./main dictionnaire_4.txt
==2793==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
a
b
d
e
f
g
r
s
y
==2793==
==2793== HEAP SUMMARY:
==2793==    in use at exit: 0 bytes in 0 blocks
==2793==   total heap usage: 15 allocs, 15 frees, 1,938 bytes allocated
==2793==
==2793== All heap blocks were freed -- no leaks are possible
==2793==
==2793== For counts of detected and suppressed errors, rerun with: -v
==2793== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

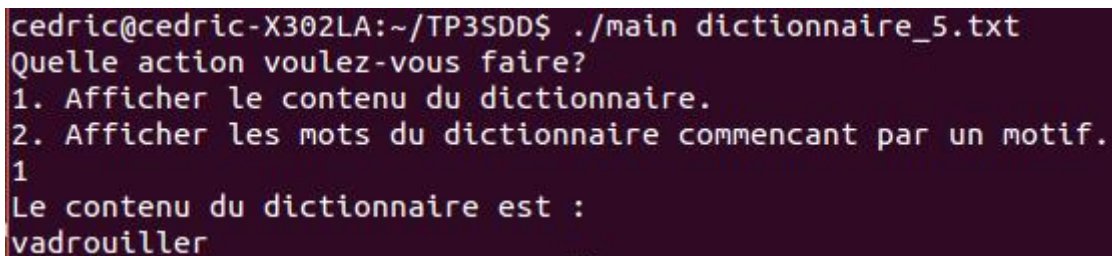
### 3.2.1.5 Lorsque fichier ne contient qu'un mot d'au 2 lettres

Les données d'entrées sont dans le fichiers dictionnaire\_5.txt



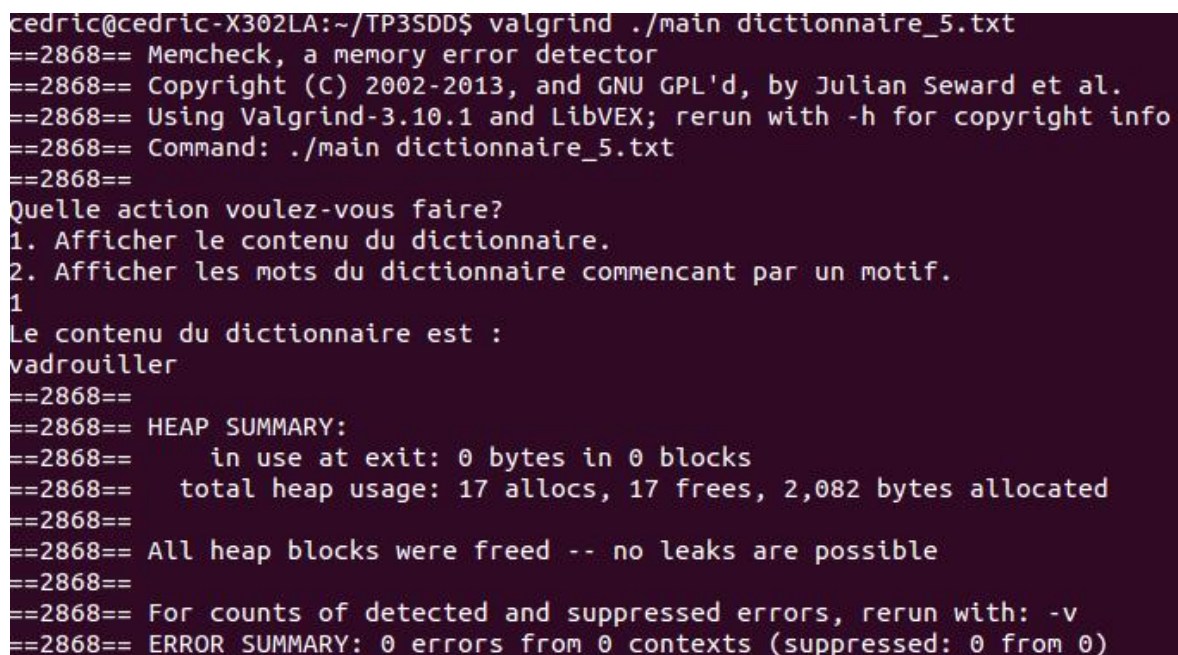
vadrouiller

Résultat de l'exécution



```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_5.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
vadrouiller
```

L'exécution avec valgrind



```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_5.txt
==2868== Memcheck, a memory error detector
==2868== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2868== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2868== Command: ./main dictionnaire_5.txt
==2868==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
vadrouiller
==2868==
==2868== HEAP SUMMARY:
==2868==   in use at exit: 0 bytes in 0 blocks
==2868==   total heap usage: 17 allocs, 17 frees, 2,082 bytes allocated
==2868==
==2868== All heap blocks were freed -- no leaks are possible
==2868==
==2868== For counts of detected and suppressed errors, rerun with: -v
==2868== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

### 3.2.1.6 Lorsque fichier ne contient au moins deux mots d'au moins deux lettres et toutes distinctes

Les données d'entrées sont dans le fichiers dictionnaire\_6.txt

```
exporter
colibacille
cocotier
bagagiste
balance
arrondissement
tarverne
vadrouiller
zoomorphe
oriental
mollet
lettrine
monoplace
hispanique
fossile
distinguer
cierge
semence
```

Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_6.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
arrondissement
bagagiste
balance
cierge
cocotier
colibacille
distinguer
exporter
fossile
hispanique
lettrine
mollet
monoplace
oriental
semence
tarverne
vadrouiller
zoomorphe
```



## 🚩 L'exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_6.txt
==2931== Memcheck, a memory error detector
==2931== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==2931== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==2931== Command: ./main dictionnaire_6.txt
==2931==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
arrondissement
bagagiste
balance
cierge
cocotier
colibacille
distinguer
exporter
fossile
hispanique
lettrine
mollet
monoplace
oriental
semence
tarverne
vadrouiller
zoomorphe
==2931==
==2931== HEAP SUMMARY:
==2931==    in use at exit: 0 bytes in 0 blocks
==2931==   total heap usage: 155 allocs, 155 frees, 12,018 bytes allocated
==2931==
==2931== All heap blocks were freed -- no leaks are possible
==2931==
==2931== For counts of detected and suppressed errors, rerun with: -v
==2931== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

### 3.2.1.7 Lorsque fichier contient au moins deux mots d'au moins deux lettres avec certain qui se répètent.

Les données d'entrées sont dans le fichier dictionnaire\_7.txt

```
exporter
colibacille
cocotier
bagagiste
balance
arrondissement
tarverne
vadrouiller
zoomorphe
oriental
mollet
lettrine
monoplace
hispanique
fossile
distinguer
cierge
arrondissement
exporter
semence
```

Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_7.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
arrondissement
bagagiste
balance
cierge
cocotier
colibacille
distinguer
exporter
fossile
hispanique
lettrine
mollet
monoplace
oriental
semence
tarverne
vadrouiller
zoomorphe
```

## 🚩 L'exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_7.txt
==3017== Memcheck, a memory error detector
==3017== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3017== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3017== Command: ./main dictionnaire_7.txt
==3017==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
arrondissement
bagagiste
balance
cierge
cocotier
colibacille
distinguer
exporter
fossile
hispanique
lettrine
mollet
monoplace
oriental
semence
tarverne
vadrouiller
zoomorphe
==3017==
==3017== HEAP SUMMARY:
==3017==    in use at exit: 0 bytes in 0 blocks
==3017== total heap usage: 155 allocs, 155 frees, 12,018 bytes allocated
==3017==
==3017== All heap blocks were freed -- no leaks are possible
==3017==
==3017== For counts of detected and suppressed errors, rerun with: -v
==3017== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

### 3.2.1.8 Lorsque fichier contient des mots d'au moins une lettre.

Les données d'entrées sont dans le fichier dictionnaire\_8.txt

```
z
exporter
a
colibacille
cocotier
bagagiste
balance
f
arrondissement
tarverne
vadrouiller
zoomorphe
e
d
oriental
mollet
lettrine
monoplace
hispanique
b
fossile
distinguer
cierge
arrondissement
exporter
semence
p
ordinateur
cahier
cartable
maison
o
a
```

```
m
entreprise
zoo
feuille
ratisser
pâtisserie
bouteille
incompatible
dossier
allonger
y
z
p
contrat
x
insurmontable
hypertrophie
kangourou
justaposition
atterage
audition
cartable
```

## 🚩 Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDB$ ./main dictionnaire_8.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
a
allonger
arrondissement
atterage
audition
b
bagagiste
balance
bouteille
cahier
cartable
cierge
cocotier
colibacille
contrat
d
distinguer
dossier
e
entreprise
exporter
f
feuille
fossile
hispanique
hypertrophie
incompatible
insurmontable
justaposition
kangourou
lettrine
m
maison
mollet
monoplace
o
ordinateur
oriental
p
pâtisserie
r
ratisser
semence
tarverne
vadrouiller
x
y
z
zoo
zoomorphe
```



## 🚩 L'exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_8.txt
==3137== Memcheck, a memory error detector
==3137== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3137== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3137== Command: ./main dictionnaire_8.txt
==3137==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
1
Le contenu du dictionnaire est :
a
allonger
arrondissement
atterage
audition
b
bagagiste
balance
bouteille
cahier
cartable
cierge
cocotier
colibacille
contrat
d
distinguer
dossier
e
entreprise
exporter
f
feuille
fossile
hispanique
hypertrophie
incompatible
insurmontable
justaposition
kangourou
lettrine
m
maison
mollet
monoplace
o
ordinateur
oriental
p
pâtisserie
r
ratisser
semence
tarverne
vadrouiller
x
y
z
zoo
zoomorphe
==3137==
==3137== HEAP SUMMARY:
==3137==   in use at exit: 0 bytes in 0 blocks
==3137==   total heap usage: 310 allocs, 310 frees, 23,178 bytes allocated
==3137==
==3137== All heap blocks were freed -- no leaks are possible
==3137==
==3137== For counts of detected and suppressed errors, rerun with: -v
==3137== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

### 3.2.2 Création du dictionnaire et affichage des mots qui commencent par un motif

Pour les différents cas particuliers concernant la création et l’affichage des mots qui commencent par un motif nous utiliserons la donnée dictionnaire\_8.txt

```
z
exporter
a
colibacille
cocotier
bagagiste
balance
f
arrondissement
tarverne
vadrouiller
zoomorphe
e
d
oriental
mollet
lettrine
monoplace
hispanique
b
fossile
distinguer
cierge
arrondissement
exporter
semence
p
ordinateur
cahier
cartable
maison
o
a
```

```
m
entreprise
zoo
feuille
ratisser
pâtisserie
bouteille
incompatible
dossier
allonger
y
z
p
contrat
x
insurmontable
hypertrophie
kangourou
justaposition
atterage
audition
cartable
```

### 3.2.2.1 Lorsque le motif n'existe pas

🚦 Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_8.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
2
Veuillez entrer le motif à rechercher
mac
Le motif n'existe pas!
```

🚦 Exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_8.txt
==3196== Memcheck, a memory error detector
==3196== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3196== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3196== Command: ./main dictionnaire_8.txt
==3196==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
2
Veuillez entrer le motif à rechercher
mac
Le motif n'existe pas!
==3196==
==3196== HEAP SUMMARY:
==3196==     in use at exit: 0 bytes in 0 blocks
==3196==   total heap usage: 309 allocs, 309 frees, 22,892 bytes allocated
==3196==
==3196== All heap blocks were freed -- no leaks are possible
==3196==
==3196== For counts of detected and suppressed errors, rerun with: -v
==3196== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

### 3.2.2.2 Lorsque la taille du motif entrée existe

🚦 Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_8.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
2
Veuillez entrer le motif à rechercher
co
Le mots commençant par le motif sont:
cocotier
colibacille
contrat
```



### Exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_8.txt
==3237== Memcheck, a memory error detector
==3237== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3237== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3237== Command: ./main dictionnaire_8.txt
==3237==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
2
Veuillez entrer le motif à rechercher
co
Le mots commençant par le motif sont:
cocotier
colibacille
contrat
==3237==
==3237== HEAP SUMMARY:
==3237==     in use at exit: 0 bytes in 0 blocks
==3237==   total heap usage: 311 allocs, 311 frees, 23,228 bytes allocated
==3237==
==3237== All heap blocks were freed -- no leaks are possible
==3237==
==3237== For counts of detected and suppressed errors, rerun with: -v
==3237== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

#### 3.2.2.3 Lorsque le motif existe et c'est un mot du dictionnaire

### Résultat de l'exécution

```
cedric@cedric-X302LA:~/TP3SDD$ ./main dictionnaire_8.txt
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
2
Veuillez entrer le motif à rechercher
zoo
Le mots commençant par le motif sont:
zoo
zoomorphe
```

## 🚩 Exécution avec valgrind

```
cedric@cedric-X302LA:~/TP3SDD$ valgrind ./main dictionnaire_8.txt
==3279== Memcheck, a memory error detector
==3279== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==3279== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==3279== Command: ./main dictionnaire_8.txt
==3279==
Quelle action voulez-vous faire?
1. Afficher le contenu du dictionnaire.
2. Afficher les mots du dictionnaire commençant par un motif.
2
Veuillez entrer le motif à rechercher
zoo
Le mots commençant par le motif sont:
zoo
zoomorphe
==3279==
==3279== HEAP SUMMARY:
==3279==     in use at exit: 0 bytes in 0 blocks
==3279==   total heap usage: 311 allocs, 311 frees, 23,228 bytes allocated
==3279==
==3279== All heap blocks were freed -- no leaks are possible
==3279==
==3279== For counts of detected and suppressed errors, rerun with: -v
==3279== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```