ISIMA
Institut Supérieur d'Informatique de Modélisation et de leurs Applications

ISIMA 1ère ANNEE

Rapport
TP3 de Structure de Données

Anaïs Darricarrere
Nada Boutadghart

13 mars 2020

Table des matières

Partie 1

Présentation générale

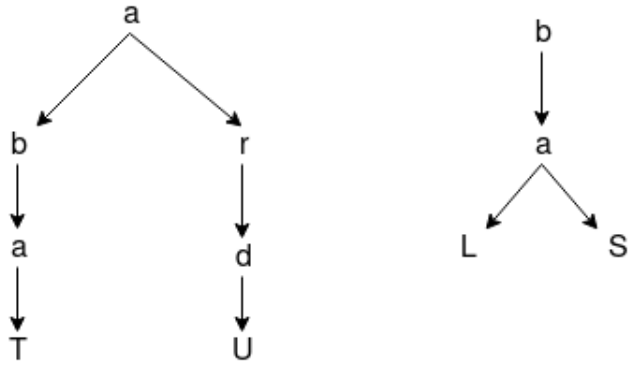
1.1 Description de l'objet du TP

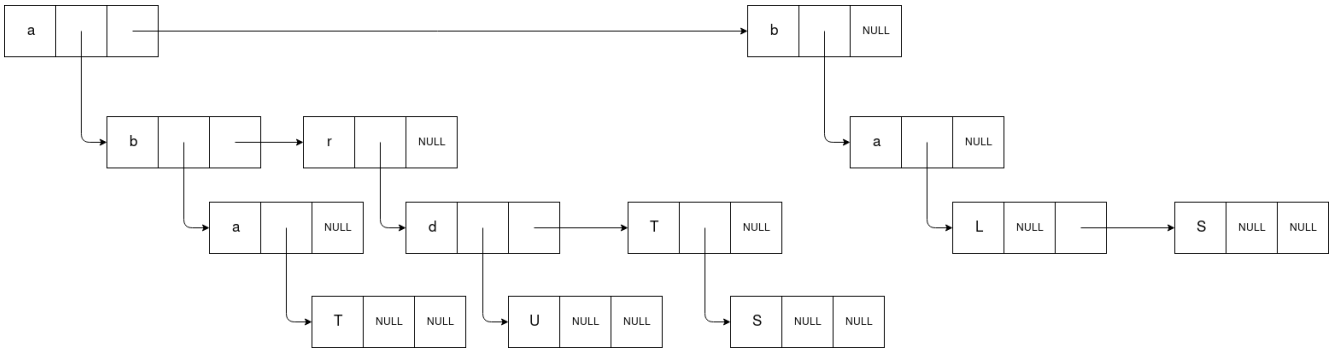
Le but de ce TP est de travailler sur la gestion d'un dictionnaire arborescent. Ce dernier contient l'ensemble de mots d'un fichier donné en entrée.

1.2 Description des structures

1.2.1 Structure d'un arbre

L'arbre fonctionne avec la structure de la liste chaînée avec des blocs de trois mots. Il est construit à partir d'un fichier d'entrée contenant la liste des mots du dictionnaire. Le premier mot contient une lettre, le deuxième mot contient le lien vertical et le troisième mot contient le lien horizontal. L'ensemble des mots est trié par ordre alphabétique sous forme de listes chaînées. Chaque lettre constitue un noeud et chaque lettre en majuscule annonce la fin d'un mot.





1.3 Organisation du code source

1.3.1 Les fichiers d'entête

Arbre.h contient :

- Des directives de préprocesseur permettant d'inclure les bibliothèques <stdio.h>, <stdlib.h>, <string.h>, <ctype.h> et "Pile.h" le fichier d'entête de la gestion de la pile écrit au TP2 ;
- La déclaration des types cellule_t ;
- Les prototypes des fonctions de gestion d'un arbre ;

1.3.2 Les modules

Arbre.c contient :

- Une directive de préprocesseur permettant d'inclure le fichier d'entête "Arbre.h" ;
- Les codes des fonctions de gestion d'un arbre déclarés dans le fichier Arbre.h sont :
  - RECH_PREC(t, mot, prof) : recherche de la chaîne de caractères mot dans le dictionnaire arborescent d'adresse de pointeur de tête t ;
  - CREER_CELL(car) : création d'une cellule avec la lettre car ;
  - ADJ_CELL(prec, nouv) : insertion d'un bloc d'adresse nouv à l'adresse prec ;
  - INSERT(t, mot) : insertion du mot dans le dictionnaire arboresent d'adresse de pointeur de tête t ;
  - LECTURE(fichier, t) : création d'un arbre de pointeur de tête t à partir d'un fichier d'entrée ;
  - AFFICHAGE(t, deb) : affichage de l'arbre en ajoutant deb comme préfixe de chaque mot ;
  - RECH_MOTIF(t, motif) : affichage de tous les mots du dictionnaire commençant par un motif donné. Pour cette fonction, on fait l'hypothèse que la recherche du motif vide ne renvoie aucun mot ;
  - SUPP_CELL(prec) : suppression de la cellule pointée par le précédent prec
  - SUPP_ARBRE(t) : suppression de l'arbre de pointeur de tête t.

Main.c contient :

- Des directives de préprocesseur permettant d'inclure les fichiers d'entête "Arbre.h" ;
- Le code du programme principal qui teste les fonctions de gestions de l'arbre ;

Partie 2

Détails des codes sources de chaque fichier

2.1 Pile.h

```c
/*-------------------------------------------------------------------------------------------------------------*/
/*                                  TP3 - Gestion d'une pile et d'une file                                     */
/*                                      Fichier d'entête Pile.h                                                */
/*                                                                                                             */
/* Déclaration des structures et des fonctions                                                                 */
/*-------------------------------------------------------------------------------------------------------------*/
#ifndef PILE_H
#define PILE_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Arbre.h"
#define NP 30               /* taille de la pile */

/*-------------------------------------------------------------------------------------------------------------*/
/* Déclaration du type d'element contenu dans la pile                                                          */
/*-------------------------------------------------------------------------------------------------------------*/

struct cellule;
typedef struct cellule * element_pile;

/*-------------------------------------------------------------------------------------------------------------*/
/* Déclaration de la structure pile avec bloc de tête                                                          */
/*      taille_max      taille de la pile                                                                      */
/*      rg_sommet       rang du sommet de la pile                                                              */
/*      ad_pile         pointeur sur la pile                                                                   */
/*-------------------------------------------------------------------------------------------------------------*/

typedef struct pile
{
    int taille_max, rg_sommet;
    element_pile * ad_pile;
}pile_t;
```

```c
/*-------------------------------------------------------------------------------------------------------------*/
/* Déclaration des fonctions                                                                                   */
/*-------------------------------------------------------------------------------------------------------------*/

pile_t * INIT_PILE (int N);

void LIBERER_PILE(pile_t * p);

int PILE_VIDE(pile_t * p);

int PILE_PLEINE(pile_t * p);

int EMPILER(pile_t* p, element_pile element);

element_pile DEPILER(pile_t *p);

element_pile SOMMET(pile_t *p);

void AFF_PILE(pile_t * p, char * deb);

#endif
```

2.2 Pile.c

```c
/*--------------------------------------------------------------------------------------*/
/*                      TP3 - Gestion d'une pile et d'une file                          */
/*                                        Pile.c                                        */
/*--------------------------------------------------------------------------------------*/

#include "Pile.h"

/*--------------------------------------------------------------------------------------*/
/* INIT_PILE               Initialisation d'une pile                                    */
/*                                                                                      */
/*  Algorithme de principe                                                              */
/*                          - Allouer le bloc de tête de la pile                        */
/*                          - Si l'allocation est réussie:                              */
/*                                  - on initialise les éléments du bloc de tête        */
/*                                  - on alloue le bloc pile                            */
/*                                  - Si l'allocation n'est pas réussie, on libère le   */
/*                                  bloc de tête                                        */
/*                          FIN                                                         */
/*                                                                                      */
/*  Lexique                                                                             */
/*                                                                                      */
/*  En entrée : N           Taille de la pile                                           */
/*                                                                                      */
/*  En sortie : p           Pointeur sur le bloc de tête de la pile                     */
/*--------------------------------------------------------------------------------------*/

pile_t * INIT_PILE (int N)
{
        pile_t * p;
        p = (pile_t*)malloc(sizeof(pile_t));
        if (p != NULL)
        {
            p->taille_max = N;
            p->rg_sommet = -1;
            p->ad_pile = (element_pile *)malloc(sizeof(element_pile)*N);
            if (p->ad_pile == NULL)
                {
                    free(p);
                    p = NULL;
                }
        }
        return p;
}
```

```c
/*-------------------------------------------------------------------------------------*/
/* LIBERER_PILE             Libération d'une pile                                      */
/*                                                                                     */
/*  Algorithme de principe                                                             */
/*                           - Libérer la pile                                         */
/*                           - Libérer le bloc de tête de la pile                      */
/*                           FIN                                                       */
/*                                                                                     */
/*  Lexique                                                                            */
/*                                                                                     */
/*  En entrée : p            Pointeur sur le bloc de tête de la pile                   */
/*-------------------------------------------------------------------------------------*/

void LIBERER_PILE(pile_t * p)
{
    free(p->ad_pile);
    free(p);
}

/*-------------------------------------------------------------------------------------*/
/* PILE_VIDE                Détermine si la pile est vide                              */
/*                                                                                     */
/*  Lexique                                                                            */
/*                                                                                     */
/*  En entrée : p            Pointeur sur le bloc de tête de la pile                   */
/*                                                                                     */
/*  En sortie :              Retourne 1 si la pile est vide et 0 sinon                 */
/*-------------------------------------------------------------------------------------*/

int PILE_VIDE(pile_t * p)
{
    return (p->rg_sommet == -1);
}

/*-------------------------------------------------------------------------------------*/
/* PILE_PLEINE              Détermine si la pile est pleine                            */
/*                                                                                     */
/*  Lexique                                                                            */
/*                                                                                     */
/*  En entrée : p            Pointeur sur le bloc de tête de la pile                   */
/*                                                                                     */
/*  En sortie :              Retourne 1 si la pile est pleine et 0 sinon               */
/*-------------------------------------------------------------------------------------*/

int PILE_PLEINE(pile_t * p)
{
    return (p->rg_sommet == p->taille_max -1);
}
```

```
/*-------------------------------------------------------------------------------------*/
/* EMPILER                  Ajoute un élément en tête de la pile                       */
/*                                                                                     */
/*  Algorithme de principe                                                             */
/*                           - Si la pile n'est pas pleine :                           */
/*                                   - on incrémente le rang du premier élément        */
/*                                   - on insère l'élément                             */
/*                           FIN                                                       */
/*                                                                                     */
/*  Lexique                                                                            */
/*                                                                                     */
/*  En entrée : p            Pointeur sur le bloc de tête de la pile                   */
/*              element      Elément à empiler                                         */
/*                                                                                     */
/*  En sortie : ok           Retourne 1 si l'élément est empilé                        */
/*-------------------------------------------------------------------------------------*/

int EMPILER(pile_t* p, element_pile element)
{
    int ok;
    ok = !(PILE_PLEINE(p));
    if (ok)
    {
        p->rg_sommet = (p->rg_sommet)+1;
        *(p->ad_pile + p->rg_sommet) = element;
    }
    return ok;
}

/*-------------------------------------------------------------------------------------*/
/* DEPILER                  Enleve l'élément au sommet de la pile                      */
/*                                                                                     */
/*  Algorithme de principe                                                             */
/*                           - Si la pile n'est pas vide:                              */
/*                                   - on récupère l'élément à dépiler                 */
/*                                   - on décrémente le rang du premier élément        */
/*                           FIN                                                       */
/*                                                                                     */
/*  Lexique                                                                            */
/*                                                                                     */
/*  En entrée : p            Pointeur sur le bloc de tête de la pile                   */
/*                                                                                     */
/*  Variables intermédiaires :                                                         */
/*              ok           Entier valant 0 si la pile est vide et 1 sinon            */
/*                                                                                     */
/*  En sortie : res          Retourne l'élément à enlever de la pile                   */
/*-------------------------------------------------------------------------------------*/

element_pile DEPILER(pile_t *p)
{
    int ok;
    element_pile res;
    ok = !(PILE_VIDE(p));
    if (ok)
    {
        res = *(p->ad_pile + p->rg_sommet);
        p->rg_sommet = (p->rg_sommet)-1;
    }
    return res;
}
```

```
/*-------------------------------------------------------------------------------------*/
/* SOMMET                   Retourne l'élément au sommet de la pile                    */
/*                                                                                     */
/*  Algorithme de principe                                                             */
/*                           - Si la pile n'est pas vide:                              */
/*                                   - on récupère l'élément au sommet                 */
/*                           FIN                                                       */
/*                                                                                     */
/*  Lexique                                                                            */
/*                                                                                     */
/*  En entrée : p            Pointeur sur le bloc de tête de la pile                   */
/*                                                                                     */
/*  Variables intermédiaires :                                                         */
/*              ok           Entier valant 0 si la pile est vide et 1 sinon            */
/*                                                                                     */
/*  En sortie : res          Retourne l'élément à enlever de la pile                   */
/*-------------------------------------------------------------------------------------*/

element_pile SOMMET(pile_t *p)
{
    int ok;
    element_pile res = NULL;
    ok = !(PILE_VIDE(p));
    if (ok)
    {
        res = *(p->ad_pile + p->rg_sommet);
    }
    return res;
}
```

```
/*-------------------------------------------------------------------------------------------*/
/* AFF_PILE                Affiche le mot dont les adresses de chaque lettre sont            */
/*                         contenues dans la pile avec un éventuel motif de début            */
/*                                                                                           */
/*  Algorithme de principe:                                                                  */
/*                         - On initialise l'entier max au rang du sommet de la pile         */
/*                         - On initialise le courant sur la liste contigue de la pile       */
/*                         - On affiche le motif de début                                    */
/*                         - On affiche les N lettres en minuscule une par une en partant    */
/*                           du sommet de la pile                                            */
/*                         FIN                                                               */
/*                                                                                           */
/*  Lexique                                                                                  */
/*                                                                                           */
/*  En entrée : p          Pointeur sur le bloc de tête de la pile                           */
/*              deb        Motif à afficher en début de chaque mot                           */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              N          Entier indiquant le rang du sommet de la pile                     */
/*              cour       Pointeur sur la pile                                              */
/*-------------------------------------------------------------------------------------------*/

void AFF_PILE(pile_t * p, char * deb)
{
    int i, N = p->rg_sommet;
    element_pile * cour = p->ad_pile;
    printf("%s", deb);
    for (i=0; i<N + 1; i++)
    {
        printf("%c", tolower((*(cour + i))->lettre));
    }
    printf("\n");
}
```

2.3 Arbre.h

```c
/*-------------------------------------------------------------------------------------------------------*/
/*                                TP3 - Gestion d'un dictionnaire arborescent                            */
/*                                    Fichier d'entête Arbre.h                                           */
/*                                                                                                       */
/* Déclaration des structures et des fonctions                                                           */
/*-------------------------------------------------------------------------------------------------------*/

#ifndef ARBRE_H
#define ARBRE_H

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

/*-------------------------------------------------------------------------------------------------------*/
/* Déclaration de la structure cellule_t : cellule de l'arbre contenant :                                */
/*      lettre          caractère d'un mot                                                               */
/*      lv              lien vertical                                                                    */
/*      lh              lien horizontal                                                                  */
/*-------------------------------------------------------------------------------------------------------*/

typedef struct cellule
{
    char lettre;
    struct cellule * lv, * lh;
}cellule_t;

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

cellule_t ** RECH_PREC(cellule_t ** t, char * mot, int * prof);

cellule_t * CREER_CELL(char car);

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

void INSERT(cellule_t ** t, char * mot);

int LECTURE(char * nom_fichier, cellule_t ** t);

void AFFICHAGE(cellule_t ** t, char * deb);

int RECH_MOTIF(cellule_t ** t, char * motif);

void SUPP_ARBRE(cellule_t ** t);

#endif
```

2.4 Arbre.c

```
/*-------------------------------------------------------------------------------------------*/
/*                          TP3 - Gestion d'un dictionnaire arborescent                      */
/*                                          Arbre.c                                          */
/*-------------------------------------------------------------------------------------------*/


#include "Arbre.h"

/*-------------------------------------------------------------------------------------------*/
/* RECH_PREC                Retourne l'adresse du pointeur (lien vertical ou                 */
/*                          lien horizontal) à modifier pour réaliser une insertion          */
/*                                                                                           */
/* Algorithme de principe                                                                    */
/*                          - On initialise le pointeur précédent sur le pointeur de         */
/*                            tête de la liste chaînée de l'arbre ;                          */
/*                          - On initialise la profondeur de l'arbre à 0 ;                   */
/*                          - On initialise un booléen stop à "faux" ;                       */
/*                          - Tant que (stop est faux) et (fin de mot non atteint)           */
/*                            et (fin d'arbre non atteinte) faire                            */
/*                              - Si la lettre courante du mot succède celle de l'arbre      */
/*                                dans l'ordre alphabétique alors on passe au lien horizontal ; */
/*                              - Sinon                                                      */
/*                                  - Si la lettre courante du mot n'est pas la dernière du mot */
/*                                    et est identique à celle de l'arbre alors              */
/*                                      - On passe au lien vertical ;                        */
/*                                      - On incrémente la profondeur ;                      */
/*                                      - On avance dans le mot ;                            */
/*                                  - Sinon on affecte "vrai" au booléen stop ;              */
/*                          FIN                                                              */
/*                                                                                           */
/* Lexique                                                                                   */
/*                                                                                           */
/*  En entrée : t           Pointeur sur le pointeur de tête de l'arbre                      */
/*              mot         Pointeur sur la chaîne de caractère contenant le mot à insérer   */
/*              prof        Adresse de l'entier indiquant la profondeur courante dans l'arbre */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              cour        Pointeur sur le bloc courant dans l'arbre                        */
/*              cour_m      Pointeur sur le courant dans le mot à insérer                    */
/*              stop        Booléen : condition d'arrêt                                      */
/*                                                                                           */
/*  En sortie : prec        Pointeur sur le précédent de la cellule recherchée               */
/*-------------------------------------------------------------------------------------------*/
```

```c
cellule_t ** RECH_PREC(cellule_t ** t, char * mot, int * prof)
{
    char * cour_m = mot;
    cellule_t ** prec = t;
    cellule_t * cour = *t;
    *prof = 0;
    int stop = 0;
    while ((cour != NULL) && (!stop) && (*cour_m != '\0'))
    {
        while ((cour != NULL) && (*cour_m > tolower(cour->lettre)))
        {
            prec = &(cour->lh);
            cour = *prec;
        }
        if ((cour != NULL) && (*cour_m == tolower(cour->lettre)) && (*(cour_m + 1) != '\0'))
        {
            prec = &(cour->lv);
            cour = *prec;
            *prof = *prof + 1;
        }
        else
            stop = 1;
        cour_m = cour_m + 1;
    }
    return prec;
}
/*-------------------------------------------------------------------------------------------*/
/* CREER_CELL               Création d'une cellule de type cellule_t                         */
/*                                                                                           */
/*  Algorithme de principe:                                                                  */
/*                           - Allouer un bloc de type cellule_t                             */
/*                           - Si l'allocation est réussie :                                 */
/*                               - Copier le caractère entré en paramètre dans la            */
/*                                 nouvelle cellule                                          */
/*                               - Initialiser les liens vertical et horizontal de la nouvelle */
/*                                 cellule à NIL                                             */
/*                           FIN                                                             */
/*                                                                                           */
/*  Lexique                                                                                  */
/*                                                                                           */
/*  En entrée : car          Caractère à mettre dans la nouvelle cellule à créer             */
/*                                                                                           */
/*  En sortie : nouv         Pointeur sur la cellule créée                                   */
/*-------------------------------------------------------------------------------------------*/

cellule_t * CREER_CELL(char car)
{
    cellule_t * nouv = NULL;
    nouv = (cellule_t*)malloc(sizeof(cellule_t));
    if (nouv != NULL)
    {
        nouv->lettre = car;
        nouv->lv = NULL;
        nouv->lh = NULL;
    }
    return nouv;
}
```

```
/*-------------------------------------------------------------------------------------------*/
/* ADJ_CELL                 Insère une cellule dans le dictionnaire arborescent à une adresse */
/*                          donnée                                                           */
/*                                                                                           */
/* Algorithme de principe                                                                    */
/*                          - Faire pointer le lien horizontal de la nouvelle cellule sur    */
/*                            la cellule suivante                                            */
/*                          - Initialiser le lien vertical de la nouvelle cellule à NIL      */
/*                          - Faire pointer la cellule précédente (lien vertical ou horizontal */
/*                            selon les cas) sur la nouvelle cellule                         */
/*                          FIN                                                              */
/*                                                                                           */
/* Lexique                                                                                   */
/*                                                                                           */
/*  En entrée : prec        Pointeur sur le précédent de l'emplacement auquel il faut insérer */
/*                          la cellule                                                       */
/*              nouv        Pointeur sur la nouvelle cellule                                 */
/*-------------------------------------------------------------------------------------------*/

void ADJ_CELL(cellule_t ** prec, cellule_t * nouv)
{
    nouv->lh = *prec;
    nouv->lv = NULL;
    *prec = nouv;
}

/*-------------------------------------------------------------------------------------------*/
/* INSERT                   Insère un mot par ordre alphabétique dans le dictionnaire        */
/*                          arborescent                                                      */
/*                                                                                           */
/* Algorithme de principe                                                                    */
/*                          - On recherche du précédent pour l'insertion du mot dans l'arbre */
/*                            (sous-programme RECH_PREC)                                     */
/*                          - Tant que la dernière lettre du mot n'est pas atteinte          */
/*                                 et elle n'est pas en majuscule faire                      */
/*                             - On insère une cellule contenant la lettre courante du mot à */
/*                               insérer dans l'arbre au niveau du précédent                 */
/*                               (sous-programme ADJ_CELL)                                   */
/*                             - On passe au lien vertical dans l'arbre                      */
/*                          - Mettre le caractère contenue dans la cellule courante en majuscule*/
/*                          FIN                                                              */
/*                                                                                           */
/* Lexique                                                                                   */
/*                                                                                           */
/*  En entrée : t           Pointeur sur le pointeur de tête de l'arbre                      */
/*              mot         Pointeur sur la chaîne de caractère contenant le mot à insérer   */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              prec        Pointeur sur le précédent du bloc courant dans l'arbre           */
/*              cour_m      Pointeur sur le courant dans le mot à insérer                    */
/*              prof        Entier indiquant la profondeur courante dans l'arbre             */
/*              nouv        Pointeur sur la cellule insérée                                  */
/*-------------------------------------------------------------------------------------------*/
```

```c
void INSERT(cellule_t ** t, char * mot)
{
    cellule_t ** prec;
    cellule_t * nouv;
    char * cour_m;
    int prof = 0;
    prec = RECH_PREC(t, mot, &prof);
    cour_m = mot + prof;
    while (*(cour_m + 1) != '\0')
    {
        nouv = CREER_CELL(*cour_m);
        if (nouv != NULL)
        {
            ADJ_CELL(prec, nouv);
            prec = &((*prec)->lv);
            cour_m = cour_m + 1;
        }
    }
    if (*prec == NULL)
    {
        nouv = CREER_CELL(*cour_m);
        if (nouv != NULL)
            ADJ_CELL(prec, nouv);
    }
    if (*prec != NULL)
        (*prec)->lettre = toupper((*prec)->lettre);
}

/*-------------------------------------------------------------------------------------------*/
/* LECTURE                      Création d'un dictionnaire arborescent à partir du fichier   */
/*                              d'entrée                                                     */
/*                                                                                           */
/*  Algorithme de principe:                                                                  */
/*                              - Initialiser une chaîne de caractère vide de 30 mots        */
/*                                (variable auxilière)                                       */
/*                              - Ouvrir le fichier en lecture                               */
/*                              - Si le fichier est ouvert :                                 */
/*                                  - Répéter :                                              */
/*                                      - Stocker le mot lu dans le fichier dans la variable */
/*                                        auxilière                                          */
/*                                      - Insérer la chaîne de caractère dans le dictionnaire*/
/*                                        arborescent (sous-programme INSERT)                */
/*                                  jusqu'à la fin du fichier                                */
/*                              - Fermer le fichier                                          */
/*                              FIN                                                          */
/*                                                                                           */
/*  Lexique                                                                                  */
/*                                                                                           */
/*  En entrée : nom_fichier     Nom du fichier d'entrée (chaîne de caractères)               */
/*              t               Pointeur sur le pointeur de tête de l'arbre                  */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              fichier         Pointeur sur le fichier d'entrée                             */
/*              lg              Entier représentant la longueur de la chaîne de caractères mot*/
/*              mot             Chaîne de caractères intermédiaire contenant le mot à insérer*/
/*                                                                                           */
/*  En sortie : erreur          Entier valant 0 si le fichier est lu et 1 sinon              */
/*-------------------------------------------------------------------------------------------*/
```

```c
int LECTURE(char * nom_fichier, cellule_t ** t)
{
    FILE * fichier;
    int erreur = 1;
    int lg;
    char mot[30];
    fichier = fopen(nom_fichier, "r");
    if (fichier != NULL)
    {
        erreur = 0;
        while(!feof(fichier))
        {
            fgets(mot, 30, fichier);
            lg = strlen(mot);
            if(lg >= 1)
            {
                if(mot[lg-1] == '\n')
                    lg = lg - 1;
                mot[lg] = '\0';
                INSERT(t, mot);
            }
        }
        fclose(fichier);
    }
    return erreur;
}
```

```
/*-------------------------------------------------------------------------------------------*/
/* AFFICHAGE                Affiche un dictionnaire arborescent déjà créé avec éventuellement */
/*                          un motif en début de chaque mot                                  */
/*                                                                                           */
/*  Algorithme de principe:                                                                  */
/*                          - On initialise le courant sur le pointeur de tête de la liste   */
/*                            chaînée                                                        */
/*                          - On initialise la pile                                          */
/*                          - Tant que la pile n'est pas vide ou qu'on est pas à la fin d'une */
/*                            branche faire                                                  */
/*                              - On empile l'adresse de la case lettre de la cellule        */
/*                              - Si la lettre est en majuscule alors                        */
/*                                  - On affiche le contenu de la pile précédée d'un éventuel */
/*                                    motif (sous-programme AFF_PILE)                        */
/*                              - On passe au lien vertical dans l'arbre                     */
/*                              - Tant qu'on est à la fin d'une branche et que la pile n'est */
/*                                pas vide faire                                             */
/*                                    - On dépile et on affecte au courant l'élément dépilé  */
/*                                    - On passe au lien horizontal dans l'arbre             */
/*                          - Libérer la pile                                                */
/*                          FIN                                                              */
/*                                                                                           */
/*  Lexique                                                                                  */
/*                                                                                           */
/*  En entrée : t               Pointeur sur le pointeur de tête de la liste chaînée         */
/*              deb             Chaîne de caractère undiquant un éventuel motif à répéter en */
/*                              début de chaque mot du dictionnaire                          */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              cour            Pointeur sur une cellule de la liste chaînée                 */
/*              p               Pointeur sur le bloc de tête de la pile                      */
/*                                                                                           */
/*  Constante globale :                                                                      */
/*              NP              Taille maximum de la pile                                    */
/*-------------------------------------------------------------------------------------------*/

void AFFICHAGE(cellule_t ** t, char * deb)
{
    cellule_t * cour = * t;
    pile_t * p;
    p = INIT_PILE(NP);
    while (!(PILE_VIDE(p)) || (cour != NULL))
    {
        EMPILER(p,cour);
        if (isupper(cour->lettre))
        {
            AFF_PILE(p, deb);
        }
        cour = cour->lv;
        while ((cour == NULL) && !(PILE_VIDE(p)))
        {
            cour = DEPILER(p);
            cour = cour->lh;
        }
    }
}
```

```c
/*-------------------------------------------------------------------------------------------*/
/* RECH_MOTIF              Affiche tous les mots d'un dictionnaire commençant par un motif   */
/*                         donné                                                             */
/*                                                                                           */
/*  Algorithme de principe:                                                                  */
/*                         - On initialise le courant sur le pointeur de tête de l'arbre     */
/*                         - On initialise la profondeur de l'arbre à 0 ;                    */
/*                         - On applique la recherche du précédent sur le motif              */
/*                           (sous-programme RECH_PREC)                                      */
/*                         - Si la lettre trouvée par la recherche du précédent correspond à */
/*                           la dernière lettre du motif alors                               */
/*                             - Si le motif est un mot existant dans le dictionnaire alors  */
/*                                 - Afficher le motif                                       */
/*                             - Afficher le contenu de l'arbre ayant pour racine la cellule */
/*                               trouvée par la recherche du précédent                       */
/*                         FIN                                                               */
/*                                                                                           */
/*  Lexique                                                                                  */
/*                                                                                           */
/*  En entrée : t              Pointeur sur le pointeur de tête de l'arbre                   */
/*              motif          Chaîne de caractère contenant le motif recherché              */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              cour           Pointeur sur le pointeur courant de l'arbre                   */
/*              prof           Entier indiquant la profondeur courante dans l'arbre          */
/*                                                                                           */
/*  Constante globale :                                                                      */
/*              NP             Taille maximum de la pile                                     */
/*                                                                                           */
/*  En sortie : ok             Entier valant 1 si le motif est présent, 0 sinon              */
/*-------------------------------------------------------------------------------------------*/

int RECH_MOTIF(cellule_t ** t, char * motif)
{
    int ok = 0;
    cellule_t ** cour = t;
    int prof = 0;
    cour = RECH_PREC(t, motif, &prof);
    if (tolower((*cour)->lettre) == motif[prof])
    {
        if (isupper((*cour)->lettre))
            printf("%s \n", motif);                /* Affichage du motif */
        AFFICHAGE(&((*cour)->lv), motif);
        ok = 1;
    }
    return ok;
}
```

```
/*-------------------------------------------------------------------------------------------*/
/* SUPP_ARBRE              Libère l'arbre                                                    */
/*                                                                                           */
/*  Algorithme de principe:                                                                  */
/*                         - On initialise le courant sur le pointeur de tête de l'arbre     */
/*                         - On initialise la pile                                           */
/*                         - Tant que la pile n'est pas vide ou qu'on est pas à la fin d'une */
/*                           branche faire                                                   */
/*                             - On empile l'adresse de la cellule courante de l'arbre       */
/*                             - On passe au lien vertical dans l'arbre                      */
/*                             - Tant qu'on est à la fin d'une branche et que la pile n'est  */
/*                               pas vide faire                                              */
/*                                     - On dépile et on affecte au courant l'élément dépilé */
/*                                     - On sauvegarde le courant dans une variable          */
/*                                       auxiliaire temp                                     */
/*                                     - On passe au lien horizontal dans l'arbre            */
/*                                     - On libère la cellule d'adresse sauvegardée dans temp*/
/*                         - Libérer la pile                                                 */
/*                         FIN                                                               */
/*  Lexique                                                                                  */
/*                                                                                           */
/*  En entrée : t          Pointeur sur le pointeur de tête de l'arbre                       */
/*                                                                                           */
/*  Variables intermédiaires :                                                               */
/*              cour       Pointeur sur la cellule courante de l'arbre                       */
/*              temp       Variable auxiliaire                                               */
/*              p          Pointeur sur le bloc de tête de la pile                           */
/*-------------------------------------------------------------------------------------------*/
void SUPP_ARBRE(cellule_t ** t)
{
    cellule_t * cour = *t;
    cellule_t * temp = NULL;
    pile_t * p;
    p = INIT_PILE(NP);
    while (!(PILE_VIDE(p)) || (cour != NULL))
    {
        EMPILER(p,cour);
        cour = cour->lv;
        while ((cour == NULL) && !(PILE_VIDE(p)))
        {
            cour = DEPILER(p);
            temp = cour;
            cour = cour->lh;
            free(temp);
        }
    }
    LIBERER_PILE(p);
}
```

2.5 Main.c

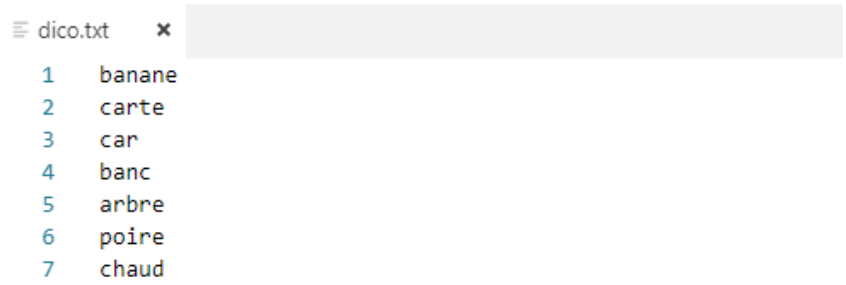
```c
/*-------------------------------------------------------------------------------------------*/
/*                          TP3 - Gestion d'un dictionnaire arborescent                      */
/*                                          Main.c                                           */
/*-------------------------------------------------------------------------------------------*/

#include "Pile.h"
#include "Arbre.h"

int main()
{
    cellule_t * t = NULL;
    cellule_t * t_vide = NULL;
    cellule_t * t_inex = NULL;

    // Création d'un arbre à partir d'un fichier vide :
    LECTURE("vide.txt", &t_vide);
    //Création d'un arbre à partir d'un fichier inexistant :
    LECTURE("inexistant.txt", &t_inex);
    // Création d'un arbre à partir d'un fichier non vide :
    LECTURE("dico.txt", &t);

    // Affichage des mots d'un dictionnaire vide :
    printf("Mots d'un dictionnaire vide : \n");
    AFFICHAGE(&t_vide, "");
    printf("\n");
    // Affichage des mots d'un dictionnaire non vide :
    printf("Mots du dictionnaire non vide : \n");
    AFFICHAGE(&t, "");
    printf("\n");

    // Recherche de mots commençant par un motif présent dans l'arbre :
    printf("Mots commençant par 'ca' : \n");
    RECH_MOTIF(&t, "ca");
    printf("\n");
    // Recherche de mots commençant par un motif inexistant dans l'arbre :
    printf("Mots commençant par 'mo' : \n");
    RECH_MOTIF(&t, "mo");
    printf("\n");
    // Recherche de mots commençant par le motif vide :
    printf("Mots commençant par le motif vide : \n");
    RECH_MOTIF(&t, "");
    printf("\n");

    // Libération de l'arbre :
    SUPP_ARBRE(&t);

    return 0;
}
```

Partie 3

Compte rendu d'éxucution

3.1 Jeux de test

Les mots à insérer dans le dictionnaire arborescent sont placés dans un fichier texte non trié. POur les jeux de tests, nous avons utilisé deux fichiers : dico.txt (joint ci-dessous) et vide.txt, le fihcier vide.

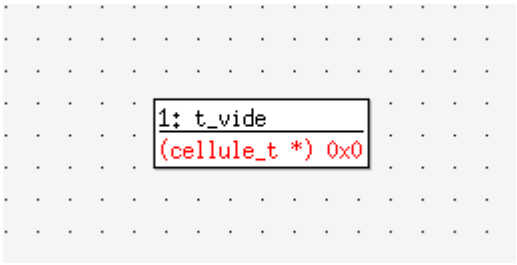


Les résultats des différents tests sont observés à travers le terminal et le débogueur ddd pour l'affichage de l'arbre.

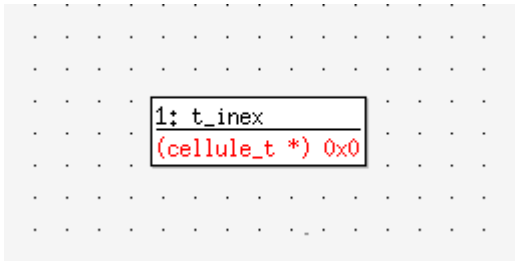
- Création d'un arbre à partir d'un fichier vide
- Création d'un arbre à partir d'un fichier inexistant
- Création d'un arbre à partir d'un fichier non vide
- Adjonction d'un mot inexistant dans l'arbre
- Adjonction d'un mot dont le début est dejà présent dans l'arbre
- Adjonction d'un mot déjà présent dans l'arbre
- Adjontion d'un mot au début du dictionnaire
- Adjontion d'un mot au milieu du dictionnaire
- Adjontion d'un mot à la fin du dictionnaire
- Affichage des mots d'un dictionnaire vide
- Affichage des mots d'un dictionnaire non vide
- Recherche des mots commençant par un motif inexistant dans l'arbre
- Recherche des mots commençant par un motif présent dans l'arbre
- Recherche des mots commençant par le motif vide
- Libération de l'arbre

Les résultats des tests effectués sont affichés ci-après :

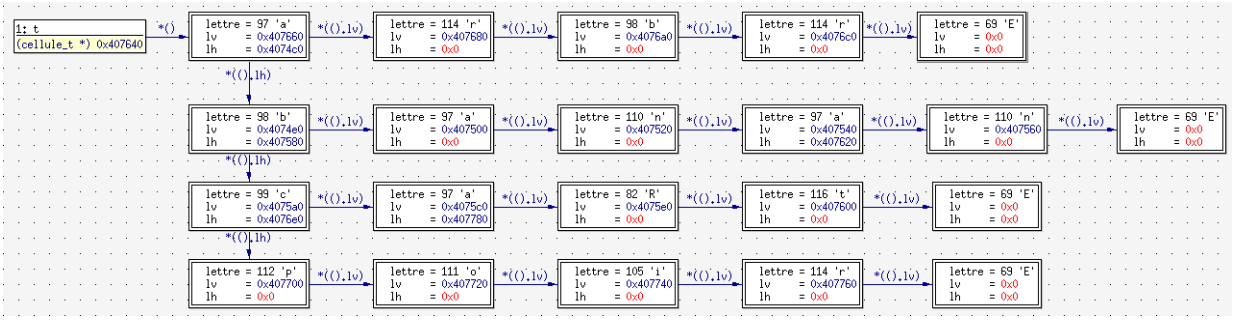
— Tests de création et adjonction de mots dans le dictionnaire arborescent :
Fichier vide



Fichier inexistant



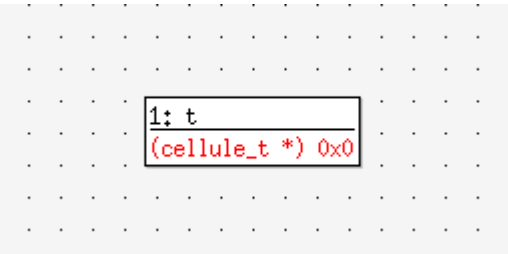
Fichier complet



— Tests d'affichage du dictionnaire arborescent et de recherche de motif :

```
[naboutadgh@etud TP3SDD]$ ./TP3
Mots d'un dictionnaire vide :

Mots du dictionnaire non vide :
arbre
banane
banc
car
carte
chaud
poire

Mots commençant par 'ca' :
car
carte

Mots commençant par 'mo' :

Mots commençant par le motif vide :
```

— Test libération de l'arbre



Vérification avec Valgrind :

```
==35932==
==35932== HEAP SUMMARY:
==35932==     in use at exit: 0 bytes in 0 blocks
==35932==   total heap usage: 40 allocs, 40 frees, 20,712 bytes allocated
==35932==
==35932== All heap blocks were freed -- no leaks are possible
==35932==
```

3.2 Makefile

```
CC = gcc
OBJS = Main.o Arbre.o Pile.o
ARGS = -Wextra -Wall -g
EXE = TP3

all : $(EXE)

$(EXE) : $(OBJS)
    $(CC) -o $(EXE) $(OBJS)
    @echo "Lancer le programme avec ./TP3"

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

clean :
    rm *.o
    rm TP3
```

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

```
[naboutadgh@etud TP3SDD]$ make
gcc -c Main.c -Wextra -Wall -g
gcc -o TP3 Main.o Arbre.o Pile.o
Lancer le programme avec ./TP3
```