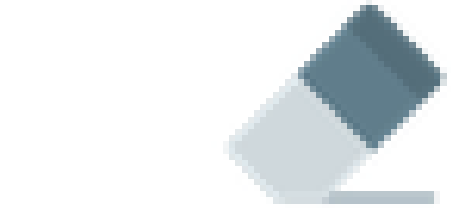
INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE



asispell

Mehdi ABOUZAID
Tanya ANGELOVA
Ali BELLAMLIH
Pierre BERNARD

Correcteur orthographique

Projet C

Algorithmique avancée - ASI - 2018

Contents

1 Introduction

1.1 Organisation

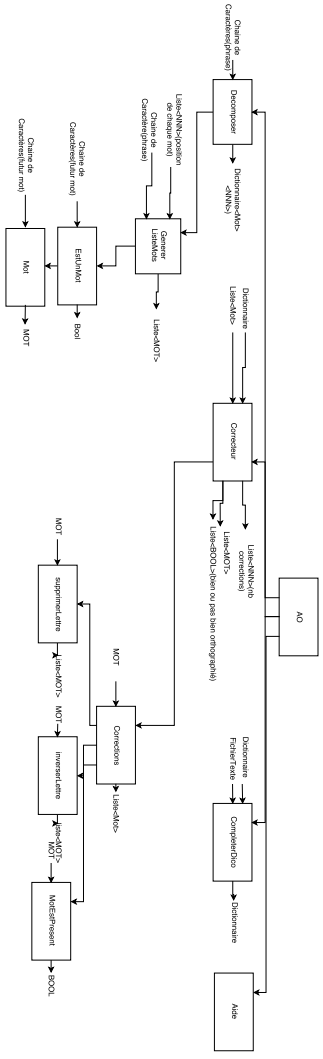
Analyse descendante : Tanya
TAD Mot : Mehdi
TAD Dictionnaire : Pierre
TAD CorrecteurOrthographique : Ali
Conception préliminaire : Pierre et Ali
Conception détaillée : tous
SDD Mot : Mehdi
SDD Dictionnaire : Pierre
SDD CorrecteurOrthographique : Ali et Tanya
Tests Mot : Mehdi
Tests dictionnaire : Mehdi
Tests CorrecteurOrthographique : Ali et Tanya

1.2 Absences

BELLAMLIH Ali le 06/12 (justifié)

2 Analyse

2.1 Analyse descendante



2.2 TAD Mot

Nom: Mot

Utilise: **Caractere**, **Chaine de caracteres**, **NaturelNonNul**, **Booleen**

Opérations:

estUnMot:	**Chaine de caracteres** $\rightarrow$ **Booleen**
mot:	**Chaine de caracteres** $\nrightarrow$ **Mot**
longueurMot:	Mot $\rightarrow$ **NaturelNonNul**
iemeCaractere:	Mot $\times$ **NaturelNonNul** $\nrightarrow$ **Caractere**
supprimerLettre:	Mot $\rightarrow$ Tableau de **Mot**
inverserLettre:	Mot $\rightarrow$ Tableau de **Mot**
motEnChaine:	Mot $\rightarrow$ **Chaine de caracteres**

Sémantiques:

estUnMot: vérifie si un mot est bien une chaine composée uniquement de lettres

mot: transforme une chaine de caractères en un Mot

supprimerLettre: génère un tableau contenant toutes les combinaisons possibles de suppression d'une lettre du Mot

inverserLettre: génère un tableau contenant toutes les combinaisons possibles d'inversion de deux lettres consécutives du Mot

motEnChaine: transforme un Mot en une chaine de caractères

Préconditions:

mot(chaine): estUnMot(chaine)

iemeCaractere(chaine,i): $1 \leq i \leq$ longueur(chaine)

2.3 TAD Dictionnaire

Nom: Dictionnaire

Paramètre:

Utilise: fichierTexte, Mot, **Booleen**, **Naturel**

Opérations: DICTIONNAIRE$_{c}ompleterDico$: Dictionnaire $\times$ FichierTexte $\rightarrow$ Dictionnaire

DICTIONNAIRE$_{m}otDansDico$: Dictionnaire $\times$ Chaîne de caractères $\rightarrow$ **Booleen**

DICTIONNAIRE$_{a}jouterDico$: Dictionnaire $\times$ Chaîne de caractères $\nrightarrow$ Dictionnaire

DICTIONNAIRE$_{d}etruireDico$: DictionnaireDictionnaireMot $\nrightarrow$ Dictionnaire

Axiomes:

Sémantiques: motEstPresent: Indique si le mot est present dans le dictionnaire

Préconditions: insererNouveauMot: non(motEstPresent(dico,mot))

supprimerMot: motEstPresent(dico,mot)

2.4 TAD CorrecteurOrthographique

Nom: CorrecteurOrthographique

Paramètre:

Utilise: Dictionnaire, Mot, **Booleen**, **Naturel**

Opérations:

correcteurOrthographique:	Dictionnaire	→ CorrecteurOrthographique
corriger:	Dictionnaire × Mot	→ Liste de mots
decomposer:	**Chaine de caracteres**	⇸ Liste de mots
nbMots:	**Chaine de caracteres**	→ **Naturel**
estCorrigeable:	**Chaine de caracteres**	→ **Booleen**

Axiomes:

Sémantiques: corriger: Permet de générer une liste de mots corrigés possibles appartenant au dictionnaire après la génération de mots aléatoires

decomposer: Génère une décomposition de la chaîne en une liste de mots contenant chaque mot de la chaîne entrée

Préconditions: corriger: non(estVide(**Chaine de caracteres**))

3 Conception préliminaire

//
fonction estUnMot (chaineATester : **Chaine de caracteres**) **: Booleen**
fonction mot (chaine:**Chaine de caracteres**) **: Mot**
⌊**précondition(s)** estUnMot(chaine)
fonction longueurMot (mot : **Mot**) **: NaturelNonNul**
fonction iemeCaractere (mot : **Mot**, position : **NaturelNonNul**) **: Caractere**
⌊**précondition(s)** $1 \leq$ position $\leq$ longueurMot(mot)
fonction supprimerLettre (mot : **Mot**) **:** Tableau de **Mot**
fonction inverserLettre (mot : **Mot**) **:** Tableau de **Mot**
fonction motEnChaine (mot : **Mot**) **: Chaine de caracteres**

fonction sousChaine (chaine : **Chaine de caracteres**, début,fin : **Naturel**) **: Chaine de caracteres**
fonction compteurDeMots (chaine : **Chaine de caracteres**) **: Naturel**
procédure decomposer (**E** ChaineADecomposer : **Chaine de caracteres**, **S** TableauDeMots : **Tableau[**1...longueurChaineADecomposer**] deChaine de caracteres**,TabPosition: **Tableau[**1...longueurChaineA**deEntier**)
fonction TrouverLesPositions (Chaine de Caractères) **:** Liste<NNN>
fonction GenererListeMots (Chaine de Caractères,Liste<NNN>) **:** Liste<MOT>
procédure correcteur (**E** Dico : **Dictionnaire**, listeMot : **Liste de Mots**, **S** nbCorrections : **Liste NaturelNonNul**, listeCorrections : **Liste de Mots**, listeValide : **Liste de booléens**)
fonction Corrections (listeEntrante:**Liste de mots**) **: Liste de mots**
fonction motEstPresent (mot : **Mot**) **: Booleen**
procédure completerDico (**E** fictexte : **FichierTexte**, **E/S** Dico :**Dictionnaire**)

4 Conception détaillée

4.1 Mot

fonction MOT_estUnMot
(chaine : **Chaine de caracteres**) **: Booleen**

Déclaration i : **NaturelNonNul**, estMot : **Booleen**, c : **Caractere**

debut
i ← 1
estMot ← VRAI
c ← chaine[1]
tant que estMot et i ≤ longueur(chaine) **faire**
c ← chaine[i]
si ((c≥'a' et c≤'z') ou (c≥'A'et c≤'Z') ou (c=='Š') ou (c=='š') ou (c=='Ž') ou (c=='ž') ou (c≥'Œ' et c≤'Ÿ') ou (c≥'À' et c≤'Ö') ou (c≥'Ø' et c≤'ö') ou (c≥'ø' et c≤'ÿ')) **alors**
sinon
i ← i+1
finsi
estMot=FAUX
fintantque
retourner estMot

fin

fonction MOT_mot (chaine : **Chaine de caracteres**) **: Mot**

Déclaration newMot : **Mot**

debut
si estUnMot(chaine) **alors**
newMot ← chaine
retourner newMot
finsi

fin

fonction MOT_longueurMot (leMot : **Mot**) **: NaturelNonNul**

Déclaration i : **NaturelNonNul**

debut
tant que leMot[i]≠" **faire**
i ← i+1
fintantque
retourner i

fin

fonction MOT_iemeCaractere (leMot : **Mot**, position:**NaturelNonNul**) : **Caractere**

debut
  si position≤longueurMot(leMot) **alors**
    retourner leMot[position]
  finsi

fin

fonction supprimerUneLettre (leMot : **Mot**, position: **NaturelNonNul**) : **Mot**

  Déclaration i : **NaturelNonNul**

debut
  pour i ←position **à** longueurMot(leMot) **faire**
    motAvecSuppression[i]=motAvecSuppression[i+1]
  finpour
  motAvecSupression[i] ← \0
  retourner motAvecSupression

fin

fonction MOT_supprimerLettre (leMot : **Mot**) : Tableau de **Mot**

  Déclaration i : **NaturelNonNul**, tableauSuppressions : Tableau de **Mot**

debut
  pour i ←1 **à** longueurMot(leMot) **faire**
    tableauSuppressions[i] ← supprimerUneLettre(leMot,i)
    i ← i+1
  finpour
  retourner tableauSuppressions

fin

fonction inverserDeuxLettres (leMot : **Mot**, position: **NaturelNonNul**) : **Mot**

debut
  si position<longueurMot(leMot) **alors**
    motAvecInversion[position] ← leMot[position+1]
    motAvecInversion[position+1] ← leMot[position]
  finsi
  retourner motAvecInversion

fin

fonction MOT_inverserLettre (leMot : **Mot**) : Tableau de **Mot**

  Déclaration i : **NaturelNonNul**, tableauInversions : Tableau de **Mot**

debut
  pour i ←1 **à** longueurMot(leMot)-1 **faire**
    tableauInversions[i] ← inverserDeuxLettres(leMot,i)
    i ← i+1
  finpour
  retourner tableauInversions

fin

fonction MOT_motEnChaine (leMot : **Mot**) **: Chaine de caracteres**

debut
  retourner leMot

fin

4.2 Correcteur Orthographique

fonction(sousChaine) : chaine: **Chaine de caracteres**, debut, fin:**NaturelNonNul Chaine de caracteres**nouveauS :**Chaine de caracteres**

nouveauS ← ""

si chaine≠"" **alors**
  pour i ←debut **à** fin **faire**
    nouveauS[i-debut] ← s[i]
  finpour

finsi

retourner nouveauS

fonction estSeparateur (car: **Caractere**) **: Booleen**

debut
  retourner (car≠(" ")) ou (car≠("'")) ou (car≠("-")

fin

fonction compteurDeMots (chaine : **Chaine de caracteres**) **: Naturel**

  Déclaration compteur,i : **Naturel**,motLu : **Booleen**

debut
  compteur ← 0
  i ← 0
  tant que i≤ longueur(chaine) **faire**
    motLu ← faux
    tant que non **estSeparateur**(**ièmeCaractère**(i)) **faire**
      i ← i+1
      motLu ← vrai

```
        fintantque
        si motLu alors
            compteur ← compteur+1
        finsi
        tant que estSeparateur(ièmeCaractère(i)) faire
            i ← i+1
        fintantque
    fintantque
    retourner compteur
fin
```

procédure decomposerLaChaine (**E** chaine **Chaine de caracteres**; **S** TableauDeMots : **Tableau[** 0..longueur(chaine)de**Chaine de caracteres] ,**TableauDePos : **Tableau[**0..longueur(chaine)**] d')**

Déclaration nbreDeMots,i,j,compteur : **Naturel**, motLu : **Booleen**, chaineADecomposer : **Chaine de caracteres**

```
debut
    i ← 0
    j ← 0
    compteur ← 0
    TableauDePos ← 0
    tant que i≤ longueur(chaine) faire
        faux ← motLu
        tant que nonestSeparateur (ièmeCaractère(i)) faire
            i ← i+1
            motLu ← vrai
        fintantque
        si motLu alors
            chaineADecomposer ← sousChaine(chaine,j,i-1)
            TableauDeMots[compteur] ← chaineADecomposer
            TableauDePos[compteur+1] ← i+1
            compteur ← compteur+1
            j ← i+1
        finsi
        tant que estSeparateur(ièmeCaractère(i)) faire
            i ← i+1
        fintantque
    fintantque
fin
```

fonction CORRestCorrigeable (mot: **Mot**, dico:**Dictionnaire**) **: Booleen**

debut

```
    si DICTIONNAIREmotDansDico(*dico, MOT_motEnChaine(mot)) alors
        retourner Faux
    sinon
        retourner Vrai
    finsi
fin
fonction CORRnbMots (str: Chaine de caracteres, dico:Dictionnaire) : Naturel
    Déclaration  mot : Mot, tableauDeMots : Tableau de Mot,res : Naturel
debut
    mot ← MOTmot(str)

    CORRcorriger(dico, mot, tableauDeMots, res )
    retourner res
fin
procédure CORRcorriger (E dico:Dictionnaire,LeMot:Mot;S tableauDeMots:Tableau[02*longeur(LeMot)-
1Mot] ,compteurDesCorrections:Natureli,j,l:Naturel,t:Tableau[02*longeur(LeMot)Mot] ,t1:Tableau[0longeur
,t2:
    )
debut
    l ← MOTlongueurMot(LeMot)
    t1 ← MOTsupprimerLettre(LeMot)
    t2 ← MOTinverserLettre(LeMot)
    *compteurDesCorrections ← 0
    pour i ←0 à l-1 faire
        t[i] ← t1[i]
    finpour
    pour i ←0 à l-1 faire
        t[i+l] ← t2[i]
    finpour
    j ← 1
    pour i ←0 à (2*l)-1 faire
        si (DICTIONNAIREmotDansDico(dico, MOTmotEnChaine(t[i]))) alors
            tableauDeMots[j] ← t[i]
            j ← j+1
        finsi
    finpour
fin
fonction correcteur (dico: textbfDictionnaire, nbMots:Naturel, tableauDesMots:Tableau[0nbMotsMot]
) : Tableau de Corrections
```

Déclaration i : **Naturel**, tableauCorrections : Tableau de **Corrections**

debut
pour i ←0 **à** nbMots **faire**
si MOTestUnMot(MOTmotEnChaine(tableauDesMots[i])) **alors**
tableauCorrections[i].mot ← tableauDesMots[i]
si CORRestCorrigeable(tableauDesMots[i], dico) **alors**
tableauCorrections[i].siMotBienEcrit ← FALSE

CORR$_c$*orriger*(*dico*, *tableauDesMots*[*i*], *tableauCorrections*[*i*].*tabCorrections*, *tableauCorrections*[*i*].*nb*
sinon
finsi
tableauCorrections[i].siMotBienEcrit ← TRUE

tableauCorrections[i].nbCorrections ← 0

tableauCorrections[i].tabCorrections[1] ← NULL
finsi
finpour
retourner tableauCorrections
fin

4.3 Dictionnaire

fonction DICTIONNAIRE_completerDico (dico : **Dictionnaire**, FichierTexte : **Chaine de caracteres**) **: Dictionnaire**

debut
pour i ←Debut FichierTexte **à** Fin FichierTExte **faire**
chaine = ligne(i)DICTIONNAIRE_ajouterDico(dico,chaine)
finpour

fin

fonction AB_ajouterArbreDroit (arbre : **AB_Arbre**, element : **caractere**) **: Dictionnaire**

debut
si (arbre est vide) **alors**
clé arbre = element
finsi
si (arbre non est vide) **alors**
obtenir fils le plus à droite de l'arbre et inserer clé arbre
finsi

fin

fonction AB_ajouterArbreGauche (arbre : **AB_Arbre**, element : **caractere**) **: Dictionnaire**

debut
  si (arbre est vide) **alors**
    clé arbre = element
  finsi
  si (arbre non est vide) **alors**
    obtenir fils le plus à droite du premier fils gauche de l'arbre et inserer clé arbre
  finsi

fin

fonction AB_ajouterArbreDroit (arbre : **AB_Arbre**, mot : **chaine de caractere**) **: booleen**

debut
  tant que dernier caractère de mot non atteint **faire**
  fintantque
  si derniere lettre de mot ET estMot(mot) **alors**
    retourner VRAI
  finsi
  si fin du dictionnaire atteint OU dernière lettre de mot atteinte et non estMot(mot) **alors**
    retourner FAUX
  finsi

fin

5 Code C et tests unitaires

5.1 Mot.h

```
/**
 * \file mot.h
 * \brief SDD mot
 * \author ABOUZAID Mehdi, ANGELOVA Tanya, BELLAMLIH Ali et BERNARD Pierre
 *
 * Implmentation des fonctions et procdures du TAD mot
 *
 */

#ifndef __MOT__
#define __MOT__


typedef char* MOT_Mot;

/**
 * \fn MOT_Mot MOT_mot(char* chaine)
 * \brief Fonction permettant de transformer une chaine en un mot de type MOT_Mot
 *
 * \param char* chaine
 * \return MOT_Mot
 */
MOT_Mot MOT_mot(char* chaine);


/**
 * \fn int MOT_estUnMot(char* chaine)
 * \brief Fonction permettant de savoir si la chaine est un mot valide ou pas
 *
 * \param char* chaine    tester
 * \return le boolen 0 pour VRAI et 1 pour FAUX
 */
int MOT_estUnMot(char* chaine);

/**
 * \fn int MOT_longueurMot(MOT_Mot leMot)
 * \brief Fonction permettant de connatre la longueur d'un mot
 *
```

```
 * \param MOT_Mot
 * \return int la longueur du mot
 */
int MOT_longueurMot(MOT_Mot leMot);

/**
 * \fn char MOT_iemeCaractere(MOT_Mot leMot, int position)
 * \brief Fonction permettant de connatre le ime caractre d'un mot
 *
 * \param MOT_Mot leMot, int position
 * \return char le ime caractre
 */
char MOT_iemeCaractere(MOT_Mot leMot, int position);


/**
 * \fn MOT_Mot* MOT_supprimerLettre(MOT_Mot leMot)
 * \brief Fonction permettant de supprimer chaque lettre d'un mot une une
 *
 * \param MOT_Mot leMot
 * \return MOT_Mot* le tableau de mots avec chacune des lettres supprimes par mot
 */
MOT_Mot* MOT_supprimerLettre(MOT_Mot leMot);

/**
 * \fn MOT_Mot* MOT_inverserLettre(MOT_Mot leMot)
 * \brief Fonction permettant d'inverser toutes les combinaisons de lettres la su
 *
 * \param MOT_Mot leMot
 * \return MOT_Mot* le tableau de mots avec les combinaisons de lettres inverses
 */
MOT_Mot* MOT_inverserLettre(MOT_Mot leMot);

/**
 * \fn char* MOT_motEnChaine(MOT_Mot leMot)
 * \brief Fonction pour transformer un mot en chaine
 *
 * \param MOT_Mot leMot
 * \return char*
 */
char* MOT_motEnChaine(MOT_Mot leMot);
```

```
#endif
```

5.2 Mot.c

```
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include"Mot.h"
#include"MotPrive.h"
#define TRUE 1
#define FALSE 0

/* Partie prive */

MOT_Mot inverserDeuxLettres(MOT_Mot leMot, int position1){
  MOT_Mot motAvecInversion=MOT_mot(leMot);
  if (position1<MOT_longueurMot(leMot)){
    motAvecInversion[position1]=leMot[position1-1];
    motAvecInversion[position1-1]=leMot[position1];
    return motAvecInversion;
  }
  else{
    return NULL;
  }
}

MOT_Mot supprimerUneLettre(MOT_Mot leMot, int position){
  int i;
  MOT_Mot motAvecSuppression=MOT_mot(leMot);
  for (i=position-1; i < MOT_longueurMot(leMot) ; i++){
      motAvecSuppression[i] = motAvecSuppression[i+1];
  }
   motAvecSuppression[i] = '\0';
  return motAvecSuppression;
}

/* Partie publique */

int MOT_estUnMot(char* chaine){
  int i=0, estMot=TRUE;
  char c=chaine[i];
```

```
  while(estMot && c!='\0' && i<strlen(chaine)){
    c=chaine[i];
    if ((c>='a' && c<='z') || (c>='A' && c<='Z') || c==' ' || c==' '|| c==' ' || c=
      i++;
    }
    else{
      estMot=FALSE;
    }
  }
  return estMot;
}

MOT_Mot MOT_mot(char* chaine){
  if (MOT_estUnMot(chaine)){
    MOT_Mot newMot=(char*)malloc(sizeof(char)*strlen(chaine));
    strcpy(newMot,chaine);
    return newMot;
  }
  else{
    return NULL;
  }
}

int MOT_longueurMot(MOT_Mot leMot){
  int i=0;
  while(leMot[i]!='\0'){
    i++;
  }
  return i;
}

char MOT_iemeCaractere(MOT_Mot leMot, int position){
  if(position<=MOT_longueurMot(leMot)){
      return leMot[position-1];
  }
  else{
    return 0;
  }
}

MOT_Mot* MOT_supprimerLettre(MOT_Mot leMot){
  int i;
```

```
  MOT_Mot* tableauSuppressions=(char**)malloc(sizeof(char)*MOT_longueurMot(leMot)*M
  for (i=0; i<MOT_longueurMot(leMot); i++){
    tableauSuppressions[i]=supprimerUneLettre(leMot,i+1);
  }
  return tableauSuppressions;
}

MOT_Mot* MOT_inverserLettre(MOT_Mot leMot){
  int i;
  MOT_Mot* tableauInversions=(char**)malloc(sizeof(char)*MOT_longueurMot(leMot)*MO
  for (i=0; i<MOT_longueurMot(leMot)-1; i++){
    tableauInversions[i]=inverserDeuxLettres(leMot,i+1);
  }
  return tableauInversions;
}

char* MOT_motEnChaine(MOT_Mot leMot){
  return leMot;
}
```

5.3 ArbreBinaire.h

```
/**
* \file ArbreBinaire.h
* \brief Implemantation du SDD Dictionnaire et des fonctions associees
* \author ABOUZAID Mehdi, ANGELOVA Tanya, BELLAMLIH Ali, BERNARD Pierre
**/

#ifndef __ARBREBINAIRE__
#define __ARBREBINAIRE__
typedef struct AB_Arbre{
    char element;
    struct AB_Arbre *filsGauche;
    struct AB_Arbre *filsDroit;
    int estMot;
} AB_Arbre;


/**
* \fn AB_ajouterArbreGauche(AB_Arbre **arbre, char element);
* \brief Fonction permettant d'ajouter un ArbreGauche    un autre arbre (le place
*
```

```
* \param Arbre, element(char)
* \return Arbre
**/
    void AB_ajouterArbreGauche(AB_Arbre **arbre, char element);

/**
* \fn AB_ajouterArbreDroit(AB_Arbre **arbre, char element);
* \brief Fonction permettant d'ajouter un ArbreGDroit    un autre arbre (le place
*
* \param Arbre, element(char)
* \return Arbre
**/
    void AB_ajouterArbreDroit(AB_Arbre **arbre, char element);

    /**
    * \fn AB_supprimerArbre(AB_Arbre **arbre);
    * \brief Fonction permettant de detruire un arbre en liberant la m moire
    *
    * \param Arbre
    * \return Arbre
    **/
    void AB_supprimerArbre(AB_Arbre **arbre);

    /**
    * \fn AB_motEstPresent(AB_Arbre *arbre, char mot[]);
    * \brief Fonction permettant de savoir si une chaine de caract re est presente
    *
    * \param Arbre, chaine de caracteres
    * \return booleen(int)
    **/
    int AB_motEstPresent(AB_Arbre *arbre, char mot[]);

    void AB_insererMot(AB_Arbre **arbre, char mot[]);

#endif
```

5.4 ArbreBinaire.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ArbreBinaire.h"
```

```
void AB_ajouterArbreDroit(AB_Arbre **arbre, char element){
    AB_Arbre *tempNoeud;
    AB_Arbre *tempArbre = *arbre;

    AB_Arbre *elem = malloc(sizeof(AB_Arbre));
    elem->element = element;
    elem->filsGauche = NULL;
    elem->filsDroit = NULL;
    elem->estMot = 0;

    if(tempArbre)
    tempNoeud = tempArbre;
    tempArbre = tempArbre->filsGauche;
    if(!tempArbre) tempNoeud->filsGauche = elem;
    if(tempArbre)
    do{
        tempNoeud = tempArbre;
        tempArbre = tempArbre->filsDroit;
        if(!tempArbre) tempNoeud->filsDroit = elem;
    }while(tempArbre);
    else  *arbre = elem;
}

/*****************************************************************************/

void AB_ajouterArbreGauche(AB_Arbre **arbre, char element){
    AB_Arbre *tempNoeud;
    AB_Arbre *tempArbre = *arbre;

    AB_Arbre *elem = malloc(sizeof(AB_Arbre));
    elem->element = element;
    elem->filsGauche = NULL;
    elem->filsDroit = NULL;
    elem->estMot = 0;

    if(tempArbre)
    do{
        tempNoeud = tempArbre;
        tempArbre = tempArbre->filsDroit;
        if(!tempArbre) tempNoeud->filsDroit = elem;
    }while(tempArbre);
```

```
    else  *arbre = elem;
}

/*************************************************************************/

void AB_supprimerArbre(AB_Arbre **arbre){
    AB_Arbre *tempArbre = *arbre;

    if(!arbre) return;

    if(tempArbre->filsGauche)  AB_supprimerArbre(&tempArbre->filsGauche);

    if(tempArbre->filsDroit) AB_supprimerArbre(&tempArbre->filsDroit);

    free(tempArbre);

    *arbre = NULL;
}

/*************************************************************************/

int AB_motEstPresent(AB_Arbre *arbre, char mot[]){
  int i = 0;
  AB_Arbre *tempArbre = arbre;

    while(i<=strlen(mot)){
    if(tempArbre->element == mot[i]){
      tempArbre = tempArbre->filsGauche;
      i++;
    }
    else
      tempArbre = tempArbre->filsDroit;

  if(tempArbre == NULL)
    return 0;
}
if(i==(strlen(mot)-1) && arbre->estMot == 1)
  return 1;

return 0;

}
```

```
void AB_insererMot(AB_Arbre **arbre, char mot[]){
  unsigned int longueurmot = strlen(mot);
  AB_Arbre *tempArbre = *arbre;

  AB_ajouterArbreDroit(&tempArbre,mot[0]);
  for(unsigned int i=1;i<longueurmot;i++){
    if(i==longueurmot-1)
      tempArbre->estMot = 1;
    AB_ajouterArbreGauche(&tempArbre,mot[i-1]);
    tempArbre = tempArbre->filsGauche;
  }
}
```

5.5 dictionnaire.h

```
/**
* \file dictionnaire.h
* \brief Implemantation du SDD Dictionnaire et des fonctions associees
* \author ABOUZAID Mehdi, ANGELOVA Tanya, BELLAMLIH Ali, BERNARD Pierre
**/

#include <stdio.h>
#include <stdlib.h>
#include "ArbreBinaire.h"

#ifndef __DICTIONNAIRE__
#define __DICTIONNAIRE__

typedef struct Dictionnaire{
  AB_Arbre Dico;
}Dictionnaire;



  /**
  * \fn void DICTIONNAIRE_completerDico(*Dictionnaire, Chaine de caracteres(nom du
  * \brief Fonction permettant de compl ter le dictionnaire
  *
  * \param Dictionnaire, FichierTexte
  * \return Dictionnaire
  **/
```

```
void DICTIONNAIRE_completerDico(Dictionnaire *dico, char FichierTexte[]);

/**
* \fn void DICTIONNAIRE_ajouterDico(*Dictionnaire, ChaineDeCaractere)
* \brief Fonction permettant de compl ter le dictionnaire
*
* \param Dictionnaire, ChaineDeCaractere
* \return Dictionnaire
**/
      void DICTIONNAIRE_ajouterDico(Dictionnaire *dico, char mot[]); // a    diter

/**
* \fn int DICTIONNAIRE_motDansDico (*Dictionnaire, ChaineDeCaractere)
* \brief Fonction permettant de compl ter le dictionnaire
*
* \param Dictionnaire, ChaineDeCaractere
* \return Booleen
**/
      int DICTIONNAIRE_motDansDico(Dictionnaire dico, char mot[]);

/**
* \fn int DICTIONNAIRE_detruireDico (*Dictionnaire)
* \brief Fonction permettant de cetruire le dictionnaire
*
* \param Dictionnaire
* \return Dictionnaire
**/
void DICTIONNAIRE_detruireDico(Dictionnaire *dico);



#endif
```

5.6 dictionnaire.c

```
#include "dictionnaire.h"
#include "ArbreBinaire.h"
#include <stdio.h>
#include <stdlib.h>

void DICTIONNAIRE_completerDico(Dictionnaire *dico, char FichierTexte[]){
```

```
    FILE *fichier = NULL;
    char chaine[40];

    fichier = fopen(FichierTexte,"r");
    if(fichier==NULL)
        ;// fichier vide(erreur)
    else{
            do{
                fgets(chaine, 40, fichier);
                if(chaine!=NULL)
                    DICTIONNAIRE_ajouterDico(dico,chaine);
            }while(chaine!=NULL);

        }
    fclose(fichier);
}

/*************************************************************************/

void DICTIONNAIRE_ajouterDico(Dictionnaire *dico, char chaine[] ){
  AB_insererMot(&dico->Dico,chaine);

}

/*************************************************************************/

int DICTIONNAIRE_motDansDico(Dictionnaire dico, char chaine[]){
  int val = AB_motEstPresent(&dico.Dico,chaine);
  return val;
}

/*************************************************************************/

void DICTIONNAIRE_detruireDico(Dictionnaire *dico){
  AB_supprimerArbre(&dico->Dico);
}
```

5.7 TADCorr.h

```
#include "Mot.h"
#include "dictionnaire.h"
#include "ArbreBinaire.h"
```

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

/**
 * \file TADCorr.h
 * \brief SDD correcteur orthographique
 * \author ABOUZAID Mehdi, ANGELOVA Tanya, BELLAMLIH Ali et BERNARD Pierre
 *
 * Implmentation des fonctions et procdures du TAD Correcteur orthographique
 *
 **/

typedef struct Correction
{
        MOT_Mot mot;
        int siMotBienEcrit;
        int nbCorrections;
        MOT_Mot tabCorrections[20];
}Correction;
/**
\fn int estSeparateur(char c);
* \brief Fonction permettant de savoir si un caractre est un sparateur de type
* \param ChaineDeCaractere, tableau de ChaineDeCaractere, Entier, tableau d'entiers
* \return tableau de ChaineDeCaractere, Entier, tableau d'entiers
**/
int estSeparateur(char c);
/**
\fn void decomposerLaChaine (const char* chaine, char*** adrPtrMots , int** adrPosD
* \brief Fonction permettant de morceler la chaine de caractre en entre pour fo
*
* \param ChaineDeCaractere, tableau de ChaineDeCaractere, Entier, tableau d'entiers
* \return tableau de ChaineDeCaractere, Entier, tableau d'entiers
**/
void decomposerLaChaine (const char* chaine,char*** adrPtrMots , int** adrPosDansC
/**
\fn int compteurDeMots (const char* chaine)
* \brief Fonction permettant de compter le nombre de mots contenu dans une chaine
*
* \param char* chaine
* \return integer
**/
int compteurDeMots (const char* chaine);
```

```
/**
\fn void char *sous_string (const char *s, unsigned int start, unsigned int end)
* \brief Fonction permettant de d couper une chaine de caract re en sous chaine d
*
* \param const char *s, unsigned int start, unsigned int end
* \return chaine de caract re
**/
char *sous_string (const char *s, unsigned int start, unsigned int end);

/**
\fn void CORR_corriger(*Dictionnaire, MOT_mot LeMot)
* \brief Fonction permettant de corriger le mot en entr e
*
* \param MOT_Mot, Dictionnaire
* \return Tableau de Mots
**/
void CORR_corriger (Dictionnaire dico, MOT_Mot LeMot, MOT_Mot *tableauDeMots, int*

/**
\fn void CORR_decomposer(*ChaineDeCaractere,*tableau de ChaineDeCaractere, Entier,
* \brief Fonction permettant de morceler la chaine de caract re en entr e pour fo
*
* \param ChaineDeCaractere, tableau de ChaineDeCaractere, Entier, tableau d'entiers
* \return tableau de ChaineDeCaractere, Entier, tableau d'entiers
**/

int CORR_nbMots(char* str, Dictionnaire dico);
/**
\fn int CORR_estCorrigeable(char str, *Dictionnaire);
* \brief Fonction permettant de verifier si une chaine est bien ecrite ou pas.
*
* \param Chaine de Caractere, Dictionnaire
* \return Boolean
**/
int CORR_estCorrigeable(MOT_Mot mot, Dictionnaire dico);
/**
\fn Correction* correcteur(Dictionnaire *dico, MOT_Mot** tableauDesMots, int nbMots
* \brief Fonction permettant de regrouper tout l'information necessaire sur chaque
*
* \param tableau, compose de tous les mots en entree, Dictionnaire, Entier - nombre
* \return un tableau de tous les mots avec tableau des ses corrections, nb de corre
**/
```

```
Correction* correcteur(Dictionnaire dico, MOT_Mot* tableauDesMots, int *nbMots);

#endif
```

5.8 CorrecteurOrthographique.c

```
#include "TADCorr.h"
#include "Mot.h"
#include <string.h>
#include "dictionnaire.h"
#define TRUE 1
#define FALSE 0
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int estSeparateur(char c){
  int a = isspace(c); // d tecte les espaces
  //int b = ispunct(c); // d tecte la ponctuation
  return ((a!=0) || (c=='-') || (c=='\'')); //(b!=0));
   }

   int compteurDeMots (const char* chaine){ //rien de sp cial    expliquer le coa
     int compteur=0;
     int i=0;
       while (i<strlen(chaine)){
         int mot_lu=0;
         while(!(estSeparateur(chaine[i]))){
           i+=1;
           mot_lu=1;
         }
         if(mot_lu) {
           compteur+=1;
         }
         while(estSeparateur(chaine[i])){
           i+=1;
         }
          //printf("%d\n",compteur);

       }
       return compteur;
```

```
    }


char *sous_string (const char *s, unsigned int start, unsigned int end){
  char *nouveau_s = NULL;

  if (s != NULL && start <= end)
  {
/* Calcul la taille de la sous chaine qui nous int resse*/
    nouveau_s = malloc (sizeof (*nouveau_s) * (end - start + 2));
    /*end-start +2 correspond    la longueur du string + 1 du '\0'*/
    if (nouveau_s != NULL)
    {
       int i;

/* On commence    parcourir s dans l'intervalle rentr  en param tre*/
       for (i = start; i <= end; i++)
       {
/* On copie dans une nouvelle chaine ce qui nous int resse */
          nouveau_s[i-start] = s[i];
       }
       nouveau_s[i-start] = '\0';
    }
   //}
    //else
    //{

    //   fprintf (stderr, "Probl me dans la d finition\n");
      // exit (EXIT_FAILURE);
   //  }
  }
  return nouveau_s;
}

void decomposerLaChaine (const char* chaine,char*** adrPtrMots , int** adrPosDans
//  char ** tabDeMots;
  int nbreDeMots= compteurDeMots(chaine);
  *adrPtrMots = malloc( sizeof(char**) * nbreDeMots ); // allouer 'compteur' * po
  *adrPosDansChaines= malloc( sizeof(int*) * nbreDeMots);
  char** ptrMots = *adrPtrMots;// utiliser un nom avec un niveau de pointeur de m
  int* PosDansChaines= *adrPosDansChaines;
  char* string_A_Afficher;
```

```
    int j=0;
    int i=0;
    int compteur=0;
    PosDansChaines[compteur]=0;
    while (i<strlen(chaine)){
      int mot_lu=0;
      while(!(estSeparateur(chaine[i]))){
        i+=1;
        mot_lu=1;
      }
        if(mot_lu) {
          char* string_A_Afficher=sous_string(chaine,j,i-1);
          ptrMots[compteur]=string_A_Afficher;
          PosDansChaines[compteur+1]=i+1;
          compteur+=1;
          //printf("%s \n",string_A_Afficher);
          j=i+1;
        }
        while(estSeparateur(chaine[i])){
          i+=1;
        }
      }
}


void CORR_corriger (Dictionnaire dico, MOT_Mot LeMot, MOT_Mot *tableauDeMots, int*
//En entree, la procedure a besoin du dictionnaire et du mot a corriger. En sortie,
        MOT_Mot *t1,*t2,*t;
        t1=(MOT_Mot*)malloc(sizeof(MOT_Mot)*MOT_longueurMot(LeMot));
        t2=(MOT_Mot*)malloc(sizeof(MOT_Mot)*(MOT_longueurMot(LeMot)-1));
        t=(MOT_Mot*)malloc(sizeof(MOT_Mot)*2*MOT_longueurMot(LeMot));
        tableauDeMots=(MOT_Mot*)malloc(sizeof(MOT_Mot)*(2*MOT_longueurMot(LeMot)-1
        int i,j;
        int l;
        l=MOT_longueurMot(LeMot);
        t1=MOT_supprimerLettre(LeMot);// nbMot=longeurDeMot
        t2=MOT_inverserLettre(LeMot);// nbMot=longueurDeMot-1
        //Concatener les deux tableaux
        *compteurDesCorrections = 0;
        for (i=0; i<=l-1; i++){
                t[i]=t1[i];
        }
```

```
        for (i=0; i<=(l-1); i++){
                t[i+l]=t2[i];
        }
        j=1;
        for (i=1; i<=((2*l)-1); i++){
                if (DICTIONNAIRE_motDansDico(dico, MOT_motEnChaine(t[i]))){
                        tableauDeMots[j]=t[i];
                        j++;

                }
        }
        *compteurDesCorrections = j;
}

int CORR_estCorrigeable(MOT_Mot mot, Dictionnaire dico){

        if (DICTIONNAIRE_motDansDico(dico, MOT_motEnChaine(mot))){
                return FALSE;
        }
        else{
                return TRUE;
        }
}
int CORR_nbMots(char* str, Dictionnaire dico){
        MOT_Mot mot;
        MOT_Mot *tableauDeMots;
        int res;
        mot = MOT_mot(str);
        CORR_corriger(dico, mot, tableauDeMots, &res );
        return res;
}

Correction* correcteur(Dictionnaire dico, MOT_Mot* tableauDesMots, int *nbMots){
        int i;
        Correction *tableauCorrections;
        tableauCorrections=(Correction*)malloc(sizeof(Correction)*(*nbMots));
        for (i=0; i<=*nbMots; i++){
                if (MOT_estUnMot(MOT_motEnChaine(tableauDesMots[i]))){
                        tableauCorrections[i].mot=tableauDesMots[i];
                        if (CORR_estCorrigeable(tableauDesMots[i], dico)){
                                tableauCorrections[i].siMotBienEcrit=FALSE;
                                CORR_corriger (dico, tableauDesMots[i], tableauCor
```

```
                        }
                        else {
                                tableauCorrections[i].siMotBienEcrit=TRUE;
                                tableauCorrections[i].nbCorrections=0;
                                tableauCorrections[i].tabCorrections[1]=NULL;
                        }
                }
        }
        return tableauCorrections;
}
```

5.9 testMot.c

```
#include <stdlib.h>
#include <CUnit/Basic.h>
#include <string.h>
#include "Mot.h"
#include"MotPrive.h"
#define TRUE 1
#define FALSE 0

int init_suite_success(void) {
  return 0;
}

int clean_suite_success(void) {
  return 0;
}

void test_mot(void){
  MOT_Mot newMot=MOT_mot("essay ");
  CU_ASSERT_TRUE(strcmp(newMot,MOT_motEnChaine("essay "))==0);
}

void test_est_un_mot(void){
  int booleen=MOT_estUnMot("     S     ");
  CU_ASSERT_TRUE(booleen);
}

void test_est_un_mot2(void){
  int booleen=MOT_estUnMot("essay&");
  CU_ASSERT_TRUE(booleen == FALSE);
```

```
}

void test_longueur_mot(void){
  MOT_Mot mot=MOT_mot("essay ");
  int longueur=MOT_longueurMot(mot);
  CU_ASSERT_TRUE(longueur==6);
}

void test_ieme_caractere(void){
  char c=MOT_iemeCaractere("essay ",6);
  CU_ASSERT_TRUE(c==' ');
}

void test_supprimer_1_lettre(void){
  MOT_Mot motSuppr=supprimerUneLettre("essay ",6);
  CU_ASSERT_TRUE(strcmp(motSuppr,MOT_motEnChaine("essay"))==0);
}

void test_inverser_2_lettres(void){
  MOT_Mot motInv=inverserDeuxLettres("essay ",1);
  CU_ASSERT_TRUE(strcmp(motInv,MOT_motEnChaine("sesay "))==0);
}

void test_mot_en_chaine(void){
  MOT_Mot mot=MOT_mot("essay ");
  CU_ASSERT_TRUE(strcmp(MOT_motEnChaine(mot),"essay ")==0);
}

void test_supprimer_lettre(void){
  MOT_Mot mot=MOT_mot("essay ");
  MOT_Mot* tableauSuppressions=MOT_supprimerLettre(mot);
  CU_ASSERT_TRUE(strcmp(tableauSuppressions[0],"ssay ")==0);
  CU_ASSERT_TRUE(strcmp(tableauSuppressions[1],"esay ")==0);
  CU_ASSERT_TRUE(strcmp(tableauSuppressions[2],"esay ")==0);
  CU_ASSERT_TRUE(strcmp(tableauSuppressions[3],"essy ")==0);
  CU_ASSERT_TRUE(strcmp(tableauSuppressions[4],"essa ")==0);
  CU_ASSERT_TRUE(strcmp(tableauSuppressions[5],"essay")==0);
}

void test_inverser_lettre(void){
  MOT_Mot mot=MOT_mot("essay ");
  MOT_Mot* tableauInversions=MOT_inverserLettre(mot);
```

```
  CU_ASSERT_TRUE(strcmp(tableauInversions[0],"sesay")==0);
  CU_ASSERT_TRUE(strcmp(tableauInversions[1],"essay")==0);
  CU_ASSERT_TRUE(strcmp(tableauInversions[2],"esasy")==0);
  CU_ASSERT_TRUE(strcmp(tableauInversions[3],"essya")==0);
  CU_ASSERT_TRUE(strcmp(tableauInversions[4],"essay")==0);
}

int main(int argc, char** argv){
  CU_pSuite pSuite = NULL;

  /* initialisation du registre de tests */
  if (CUE_SUCCESS != CU_initialize_registry())
    return CU_get_error();

  /* ajout d'une suite de test */
  pSuite = CU_add_suite("Tests boite noire", init_suite_success, clean_suite_succes
  if (NULL == pSuite) {
    CU_cleanup_registry();
    return CU_get_error();
  }

  /* Ajout des tests ? la suite de tests boite noire */
  if ((NULL == CU_add_test(pSuite, "mot", test_mot))
      || (NULL == CU_add_test(pSuite, "est un mot", test_est_un_mot))
      || (NULL == CU_add_test(pSuite, "est un mot2", test_est_un_mot2))
      || (NULL == CU_add_test(pSuite, "longueur", test_longueur_mot))
      || (NULL == CU_add_test(pSuite, "ieme caractere", test_ieme_caractere))
      || (NULL == CU_add_test(pSuite, "supprimer 1 lettre", test_supprimer_1_lettre
      || (NULL == CU_add_test(pSuite, "inverser 2 lettres", test_inverser_2_lettres
      || (NULL == CU_add_test(pSuite, "mot en chaine", test_mot_en_chaine))
      || (NULL == CU_add_test(pSuite, "tableau supprimer lettre", test_supprimer_l
      || (NULL == CU_add_test(pSuite, "tableau inverser lettre", test_inverser_lett
     )
  {
    CU_cleanup_registry();
    return CU_get_error();
  }

  /* Lancement des tests */
  CU_basic_set_mode(CU_BRM_VERBOSE);
  CU_basic_run_tests();
  printf("\n");
```

```
   CU_basic_show_failures(CU_get_failure_list());
   printf("\n\n");

   /* Nettoyage du registre */
   CU_cleanup_registry();
   return CU_get_error();
}
```

5.10 testDictionnaire.c

```
#include <stdlib.h>
#include <CUnit/Basic.h>
#include <string.h>
#include "Mot.h"
#include "dictionnaire.h"
#define TRUE 1
#define FALSE 0

int init_suite_success(void) {
   return 0;
}

int clean_suite_success(void) {
   return 0;
}

void test_completer_dico(void){
   Dictionnaire dico;
   MOT_Mot mot=MOT_mot("blabla");
   DICTIONNAIRE_completerDico(&dico,"test.txt");
   int booleen=DICTIONNAIRE_motDansDico(dico,mot);
   CU_ASSERT_TRUE(booleen)
}



void test_mot_dans_dico(void){
   Dictionnaire dico;
   MOT_Mot mot=MOT_mot("bonjour");
   DICTIONNAIRE_ajouterDico(&dico,mot);
   int booleen=DICTIONNAIRE_motDansDico(dico,mot);
   CU_ASSERT_TRUE(booleen)
```

```
}


void test_ajouter_dico(void){
  Dictionnaire dico;
  MOT_Mot mot=MOT_mot("bonjour");
  DICTIONNAIRE_ajouterDico(&dico,mot);
  int booleen=DICTIONNAIRE_motDansDico(dico,mot);
  CU_ASSERT_TRUE(booleen)
}

void test_detruire_dico(void){
  Dictionnaire dico;
  MOT_Mot mot=MOT_mot("bonjour");
  DICTIONNAIRE_detruireDico(&dico);
  int booleen=DICTIONNAIRE_motDansDico(dico,mot);
  CU_ASSERT_TRUE(booleen==FALSE)
}



int main(int argc, char** argv){
  CU_pSuite pSuite = NULL;

  /* initialisation du registre de tests */
  if (CUE_SUCCESS != CU_initialize_registry())
    return CU_get_error();

  /* ajout d'une suite de test */
  pSuite = CU_add_suite("Tests boite noire", init_suite_success, clean_suite_succes
  if (NULL == pSuite) {
    CU_cleanup_registry();
    return CU_get_error();
  }

  /* Ajout des tests ? la suite de tests boite noire */
  if ((NULL == CU_add_test(pSuite, "completer dico", test_completer_dico))
      || (NULL == CU_add_test(pSuite, "ajouter dico", test_ajouter_dico))
      || (NULL == CU_add_test(pSuite, "mot dans dico", test_mot_dans_dico))
      || (NULL == CU_add_test(pSuite, "detruire dico", test_detruire_dico))
       )
    {
```

```
      CU_cleanup_registry();
      return CU_get_error();
    }

  /* Lancement des tests */
  CU_basic_set_mode(CU_BRM_VERBOSE);
  CU_basic_run_tests();
  printf("\n");
  CU_basic_show_failures(CU_get_failure_list());
  printf("\n\n");

  /* Nettoyage du registre */
  CU_cleanup_registry();
  return CU_get_error();
}
```

5.11 TestCorrecteurOrthographique.c

```
#include <stdlib.h>
#include <CUnit/Basic.h>
#include <string.h>
#include "TADCorr.h"
#include "Mot.h"
#include "dictionnaire.h"
#define TRUE 1
#define FALSE 0

int init_suite_success(void) {
  return 0;
}

int clean_suite_success(void) {
  return 0;
}

void test_sous_string1(void) {
  char* testString = "TestMot";
  char* test = "Test";
  //int booleen = strcmp(sous_string(testString,0,3),test);
  CU_ASSERT_TRUE(test==sous_string(testString,0,3));
}
void test_sous_string2(void) { // a voir comment ca fonctionne
```

```
  char* String = "azedsq";
  char* String_faux = sous_string(String,3,0);

  CU_ASSERT_TRUE(String_faux=NULL);
//  int booleen = strcmp(String_faux,"");  je resous un conflit, je ne suis pas sure
//  CU_ASSERT_TRUE(booleen=TRUE)
}
void test_estSeparateur1(void){
  char* String ="j'aime les choux-fleurs";
  int booleen=estSeparateur(String[1]);
  CU_ASSERT_TRUE(booleen=TRUE);
}

void test_estSeparateur2(void){
  char* String ="j'aime les choux-fleurs";
  int booleen=estSeparateur(String[6]);
  CU_ASSERT_TRUE(booleen=TRUE);
}
void test_estSeparateur3(void){
  char* String ="j'aime les choux-fleurs";
  int booleen=estSeparateur(String[16]);
  CU_ASSERT_TRUE(booleen=TRUE);
}
void test_decomposer_compteurDeMots(void){
  char* String_a_decomposer = "String a decomposer";
  char ** tabMots;
  int * pos;
  int compteur=compteurDeMots(String_a_decomposer);
  decomposerLaChaine(String_a_decomposer,&tabMots,&pos);
  char* charTest=tabMots[0];
  int position_a=pos[1];
  CU_ASSERT_TRUE((strcmp(charTest,"String")==0) && (compteur==3) && (position_a==7
}

void test_estCorrigeable1(void){
        MOT_Mot mot = MOT_mot("vrai");
        Dictionnaire dico;
        DICTIONNAIRE_ajouterDico(&dico, "vrai");
//      FILE* fichier=NULL;
 //    fichier=fopen("test.txt","w");
 //    fwrite(mot, MOT_longueurMot(mot)+1, 1, fichier);
 //    Dictionnaire dico=DICTIONNAIRE_creationDictionnaire(fichier);
```

```
        CU_ASSERT_TRUE(CORR_estCorrigeable(mot, dico)==0);
}

void test_estCorrigeable2(void){
        MOT_Mot mot = MOT_mot("vrei");
//      MOT_Mot motVr = MOT_mot("vrai");
        Dictionnaire dico;
        DICTIONNAIRE_ajouterDico(&dico, "vrai");
        CU_ASSERT_TRUE(CORR_estCorrigeable(mot, dico)==1);
}

void test_corriger(void){
        MOT_Mot mot = MOT_mot("vraei");
//      MOT_Mot motVr = MOT_mot("vrai");
        MOT_Mot *tableauDesMots;
        Dictionnaire dico;
        DICTIONNAIRE_ajouterDico(&dico, "vrai");
        tableauDesMots=(MOT_Mot*)malloc(sizeof(MOT_Mot)*(2*MOT_longueurMot(mot)-1)
        int compteurDesCorrections=0;
//      FILE* fichier=NULL;
//      fichier=fopen("test.txt","w");
//      fwrite(motVr, MOT_longueurMot(motVr)+1, 1, fichier);
//      Dictionnaire dico=DICTIONNAIRE_creationDictionnaire(fichier);
        CORR_corriger (dico, mot, tableauDesMots, &compteurDesCorrections);
        CU_ASSERT_TRUE((tableauDesMots[1]==MOT_mot("vrai"))&&(compteurDesCorrection
}

void test_correcteur(void){
        int nbMots=2;
        MOT_Mot *tableauDesMots;
        tableauDesMots=(MOT_Mot*)malloc(sizeof(MOT_Mot)*nbMots);
        Correction *tableauDesCorrections;
        tableauDesCorrections=(Correction*)malloc(sizeof(Correction)*nbMots);
        Dictionnaire dico;
        DICTIONNAIRE_ajouterDico(&dico, "vrai");
//      FILE* fichier=NULL;
//      fichier=fopen("test.txt","w");
//      fwrite(MOT_mot("vrai"), MOT_longueurMot(MOT_mot("vrai"))+1, 1, fichier);
//      Dictionnaire dico=DICTIONNAIRE_creationDictionnaire(fichier);
        tableauDesMots[1]=MOT_mot("vrai");
        tableauDesMots[2]=MOT_mot("vraei");
        tableauDesCorrections = correcteur(dico, tableauDesMots,&nbMots);
```

```
        CU_ASSERT_TRUE((tableauDesCorrections[1].mot==tableauDesMots[1])&&(tableauD
        CU_ASSERT_TRUE((tableauDesCorrections[1].nbCorrections==0)&&(tableauDesCor
        CU_ASSERT_TRUE((tableauDesCorrections[2].mot==tableauDesMots[2])&&(tableauD
        CU_ASSERT_TRUE((tableauDesCorrections[1].nbCorrections==1)&&(tableauDesCor
}
int main(int argc, char** argv){
  CU_pSuite pSuite = NULL;

  /* initialisation du registre de tests */
  if (CUE_SUCCESS != CU_initialize_registry())
    return CU_get_error();

  /* ajout d'une suite de test */
  pSuite = CU_add_suite("Tests boite noire", init_suite_success, clean_suite_succes
  if (NULL == pSuite) {
    CU_cleanup_registry();
    return CU_get_error();
  }

  /* Ajout des tests ? la suite de tests boite noire */
  if ((NULL == CU_add_test(pSuite, "partie d'une chaine1", test_sous_string1))
      || (NULL == CU_add_test(pSuite, "partie d'une chaine2", test_sous_string2))
      || (NULL == CU_add_test(pSuite, "caractere est separateur1", test_estSeparate
      || (NULL == CU_add_test(pSuite, "caractere est separateur2", test_estSeparate
      || (NULL == CU_add_test(pSuite, "caractere est separateur3", test_estSeparate
      || (NULL == CU_add_test(pSuite, "Decomposer une phrase", test_decomposer_comp
      || (NULL == CU_add_test(pSuite, "mot est corrigeable1", test_estCorrigeable1)
      || (NULL == CU_add_test(pSuite, "mot est corrigeable2", test_estCorrigeable2)
      || (NULL == CU_add_test(pSuite, "corriger un mot", test_corriger))
      || (NULL == CU_add_test(pSuite, "corriger suite des mots", test_correcteur))
     )
    {
      CU_cleanup_registry();
      return CU_get_error();
    }

  /* Lancement des tests */
  CU_basic_set_mode(CU_BRM_VERBOSE);
  CU_basic_run_tests();
  printf("\n");
  CU_basic_show_failures(CU_get_failure_list());
  printf("\n\n");
```

```
  /* Nettoyage du registre */
  CU_cleanup_registry();
  return CU_get_error();
}
```

5.12 main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "TADCorr.h"
#include "Mot.h"
#include "dictionnaire.h"
#include "ArbreBinaire.h"


int main(int argc,char** argv){

int compteur,Corrections,i,nombreArguments=argc-1;
MOT_Mot* tableauDeMots;

  if ((nombreArguments==2) || (nombreArguments==4)) {
    if (strcmp(argv[1], "-h")==0) {
      //printf l'aide
    }
    else if(strcmp(argv[1], "-d")==0){
      //correction entree standard avec dico argv[2]
      char input[500];
      fgets(input,500,stdin);
      Correction *corrections;
      corrections=(Correction*)malloc(sizeof(Correction)*compteur);
      Dictionnaire dico;
      int *PositionDansChaine;
      *PositionDansChaine = malloc( sizeof(char**) * compteur );
      decomposerLaChaine (input,&tableauDeMots,&PositionDansChaine);
      compteur = compteurDeMots(input);
      Corrections=correcteur(dico,tableauDeMots,&compteurDeMots);
      for (i=0 ; i<compteur ; i++){
              printf("%s/n",corrections[i]);
        if (nombreArguments==4) {
                if (strcmp(argv[3], "-f")==0){
```

```
                    DICTIONNAIRE_completerDico(&argv[2],argv[4]);
                }
            }
        }
    }
    return 0;
}
```

6 Conclusion

Nous avons découverts le C cette année à travers les premiers TP mais ce n'est qu'au cours de ce projet que nous avons réellement pu appréhender les possibilités qu'offrait la programmation en C. De nombreux problèmes ont été rencontrés au cours de la programmation mais nous avons toujours cherché une solution. Réussir à travailler en groupe, à se répartir les tâches n'est pas aisé mais c'est grâce à ce genre de projet et aux problèmes rencontrés que nous nous améliorons afin d'être plus efficaces et autonomes.

6.1 Conclusion Mehdi

J'ai trouvé ce projet intéressant et enrichissant d'un point de vue technique et humain. Il m'a permis de connaître et savoir utiliser le logiciel git et de m'améliorer au niveau de la conception et programmation tout en essayant de surmonter les difficultés rencontrées. Concernant l'organisation et le travail en équipe, je trouve que le projet nous offre une bonne expérience pour la suite de notre cursus ASI.

6.2 Conclusion Tanya

Pour moi, ce projet était vraiment important. Le travail m'a permis de sentir l'esprit d'équipe et d'améliorer mon niveau de programmation et de latex. Nous avons chercher des solutions ensemble et en même temps nous avons réussi a bien diviser le travail pour atteindre un résultat plus rapidement. Le projet m'a permis d'apprendre beaucoup, alors je trouve qu'il est crucial pour notre avenir.

6.3 Conclusion Ali

Ce projet, malgré qu'on ait pas réussi à le finir correctement fut une bonne expérience car il nous a permit de mettre en pratique les connaissances théoriques du cours. De ce que j'ai pu constater, la maîtrise du C demande nettement plus de temps, d'effort mais surtout de patience en comparaison à un langage comme Pascal. J'ai dû passer énormément de temps à chercher sur internet et à réfléchir à la syntaxe sur des choses sur lesquelles j'aurai passé pas plus de 5 minutes sur Pascal ou sur papier. Là résidait ma principale source de peine et de contrariété dans ce projet. De plus, je pense que c'est bien la 1ère fois où je travaille avec une équipe sans aucun souci ou dispute. Il y a eu une bonne entente entre tous les membres tout au long du projet, et nous nous sommes tous donnés à fond sous la direction de notre chef Mehdi. Quelques regrets non négligeables tout de même de ne pas avoir pu faire fonctionner le dictionnaire.

6.4 Conclusion Pierre

J'ai apprecié ce travail en groupe sur un projet. Il était conséquent pour ne pas pouvoir être assurer par une seule personne et nous à donc permi d'apprendre à travailler sur un programme en équipe. De plus, la programmation sous la forme de projet est bien plus enrichissante et nous en apprend bien plus que de simples TPs avec des consignes, on doit chercher, comprendre et développer par nous

même. C'était donc très interessant au niveau de l'apprentissage, de la méthode ainsi que du travail en équipe.