

Alexis IMBERT  
Ruth LIOTÉ  
Amina SAKOUTI  
Léo ZEDEK

## Correcteur Orthographique ITI ALGO 2021 2022

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.2	Cahiers des charges . . . . .	3
1.3	Comment corriger un mot ? . . . . .	4
1.4	Travail demandé . . . . .	5
1.5	Ajout . . . . .	5
<b>2</b>	<b>Analyse</b>	<b>7</b>
2.1	Présentation des TAD . . . . .	7
2.1.1	TAD FichierTexte . . . . .	7
2.1.2	TAD Mot . . . . .	7
2.1.3	TAD Dictionnaire . . . . .	8
2.1.4	TAD CorrecteurOrthographique . . . . .	9
2.2	Analyse Descendante . . . . .	10
<b>3</b>	<b>Conception Préliminaire</b>	<b>11</b>
3.1	Signature FichierTexte . . . . .	11
3.2	Signature Mot . . . . .	11
3.3	Signature Dictionnaire . . . . .	12
3.4	Signature CorrecteurOrthographique . . . . .	12
<b>4</b>	<b>Conception Détaillé</b>	<b>13</b>
4.1	Conception Détaillé du TAD Fichier Texte . . . . .	13
4.2	Conception Détaillé du TAD mot . . . . .	13
4.3	Conception Détaillé du TAD Dictionnaire . . . . .	16
4.4	Conception Détaillé du TAD Correcteur Orthographique . . . . .	20
<b>5</b>	<b>Implémentation de la conception en C</b>	<b>22</b>
5.1	TAD mot . . . . .	22
5.1.1	Fichier mot.h . . . . .	22
5.1.2	Fichier mot.c . . . . .	22
5.1.3	Fichier testsMot.c . . . . .	22
5.2	TAD dictionnaire . . . . .	22
5.2.1	Fichier dictionnaire.h . . . . .	22
5.2.2	Fichier dictionnaire.c . . . . .	22
5.2.3	Fichier testsDictionnaire.c . . . . .	22
5.3	TAD correcteurOrthographique . . . . .	22
5.3.1	Fichier correcteurOrthographique.h . . . . .	22
5.3.2	Fichier correcteurOrthographique.c . . . . .	22
5.3.3	Fichier testsCorrecteurOrthographique.c . . . . .	22

## TABLE DES MATIÈRES

<b>6 Conclusion</b>	<b>23</b>
6.1 Conclusions personnelles . . . . .	23
6.1.1 Conclusion Alexis IMBERT . . . . .	23
6.1.2 Conclusion Ruth LIOTÉ . . . . .	23
6.1.3 Conclusion Amina SAKOUTI . . . . .	23
6.1.4 Conclusion Léo ZEDEK . . . . .	23
6.2 conclusion générale du projet . . . . .	23

# 1 Introduction

## 1.1 Contexte

Ce projet a été réalisé par Alexis IMBERT, Ruth LIOTÉ, Amina SAKOUTI et Léo ZEDEK dans le cadre du cours d'algorithmie avancé et programmation C, suivie en ITI3 et dirigé par M DELESTRE, le projet a été supervisé par M GUERIAU.

## 1.2 Cahiers des charges

L'objectif de ce projet est de développer un correcteur orthographique à l'image des programmes unix ispell et aspell 1. Les fonctionnalités attendues de ce programme sont les suivantes.

**Aider** Lorsqu'il est lancé sans option, ou avec l'option -h, ou avec une option non reconnue, le programme doit afficher une aide.

Par exemple :

```
. / a s i s p e l l
A i d e d e a s i s p e l l :
a s e s p e l l -h : c e t t e a i d e
a s i s p e l l d i c o : c o r r e c t i o n d e l 'l e n t r e e s s u t
    _ a _ n _ d _ a _ r _ d _ a _ l _ ' a i d e d u d i c t i o n n a i r e d i c o
a s i s p e l l -d d i c o -f f i c : c o m p l e t e r l e d i c t i
    o n n a i r e d i c o a l 'l a i d e d e s m o t s d u f i c h i e r
    _ f i c _ ( _ u n _ m o t _ p a r _ l i g n e _ )
```

**Compléter un dictionnaire** Il doit être possible de compléter un dictionnaire à l'aide des mots d'un fichier texte (un mot par ligne). Les options sont alors :

- -d pour spécifier le nom du fichier correspondant au dictionnaire utilisé ;
- -f pour spécifier le nom du fichier contenant les mots à ajouter.

L'exemple suivant ajoute les mots du fichier dico-ref\_ascii.txt (téléchargeable sur moodle) au dictionnaire francais.dico :

```
. / a s i s p e l l -d f r a n c a i s . d i c o -f d i c o - r e f - a s c i i . t x t
```

**Corriger** A l'image du programme ispell, le programme doit pouvoir analyser (et proposer des corrections orthographiques si besoin) un texte qui lui est donné via l'entrée standard. Cette analyse est

dépendante d'un dictionnaire (option -d). Cette analyse sera produite sur la sortie standard avec un mot analysé par ligne. Deux cas de figure sont traités :

1. un mot est bien orthographié (présent dans le dictionnaire), le programme produit une étoile (\*) pour ce mot ;
2. un mot est mal orthographié, le programme produit un et commercial (&) suivi des informations suivantes :
  - le mot mal orthographié ;
  - le nombre de corrections proposées ;
  - le position du mot mal orthographié depuis le début du flux d'entrée, suivi de deux points

— les corrections proposées séparées par un espace.

Voici un exemple d'utilisation :

```
e c h o " _ y n _ p e t t _ t _ t _ s _ e t _ d _ e _ c _ c _ o _ r _ r _ e _ c _ t _ i _ o _ n _ o _ r _ t _ h _ o _ g _ r _ a _ p _ h _ y _ q
  _ u _ e _ a _ v _ e _ c _ q _ u _ e _ l _ q _ u _ e _ s
f _ o _ u _ t _ e _ s _ d _ ' _ o _ r _ t _ h _ o _ g _ r _ a _ p _ h _ e _ . _ " | . / a s i s p e l l - d f r a n c
  a i s . d i c o
& y n 7 1 : y i n y e n u n o n i n e n a n
& p e t t 2 4 : p e t i t p e u t
& t s e t 1 1 1 : t e s t
*& c o r r e c t i o n 1 1 9 : c o r r e c t i o n
& o r t h o g r a p h y q u e 1 2 9 : o r t h o g r a p h i q u e
*
*& f o t e s 1 6 5 8 : f o u t e s f o r t e s f o n t e s f o r e s f o i
  e s f û t e s f î t e s f ê t e s
v o t e s r o t e s p o t e s n o t e s l o t e s d o t e s c o t e s b o
  t e s
& d 1 1 6 4 : d û d é d u d o d m d l d g d e à y a
*
```

### 1.3 Comment corriger un mot ?

Pour proposer des corrections d'un mot mal orthographié (pour un dictionnaire donné) le plus simple est de partir de ce mot et de lui appliquer des transformations qui permettent d'obtenir des mots proches. Si une transformation donne un mot bien orthographié alors ce mot est une correction possible. Voici quelques exemples de transformation :

- remplacement d'une lettre ;
- inversion de deux lettres consécutives ;
- suppression d'une lettre ;
- insertion d'une lettre ;
- décomposition d'un mot en plusieurs mots ;

- proposition d'un mot phonétiquement proche ;
- etc.

Dans l'exemple précédent, ce sont ces trois premières stratégies qui sont utilisées (c'est pour cela que le programme n'arrive pas à proposer une correction valable pour le mot fote).

L'inconvénient de cette méthode est qu'elle génère beaucoup de mots. Par exemple pour un mot de 5 lettres, la première stratégie va générer  $5 * (26 + 16) - 5 = 205$  mots (26 lettres de l'alphabet et 16 versions accentuées). Il faut donc que être capable de vérifier si un mot est présent dans un dictionnaire de manière efficace (le fichier dico-ref\_ascii.txt proposé contient 336531 mots)

## 1.4 Travail demandé

**Analyse** On nous donne les type Mode, et le TAD FichierTexte, dans le rapport.

Nous avons du spécifier les TADs Mot, Dictionnaire et CorrecteurOrthographique.

### Conception

**Conception préliminaire** Nous avons données les signatures des fonctions et procédures des TAD précédents ainsi que celles issues de l'analyse descendantes (fonctions et procédures métiers)

**Conception détaillée** Nous avons fais l'analyse descendantes des fonctions et procédures des TAD précédents les plus complexes. Ainsi que celle des fonctions et procédures métiers.

**Développement, tests unitaires et documentation** Développer le programme en C en testant au maximum vos fonctions à l'aide de l'API CUnit (la personne qui codera les tests unitaires ne doit pas être la personne qui implante les fonctions C). Vous générerez aussi une documentation de votre code à l'aide du logiciel Doxygen

## 1.5 Ajout

Le pdf du rapport peut être obtenue grace à la compilation du fichier .tex (make dans le dossier rapport).

Le correcteur orthographique est lui aussi compilable grace un makefile dans le dossier programme,

Les options du makefile

- all (make par défaut) compile le programme, les tests unitaires et la documentation

- tests compile seulements les tests unitaires
- doc compile uniquement la documentation Doxygen

Nous avons ajouté un script à la racine du projet qui permet de compilé à la fois le rapport et le programme (option all) en copiant dans le dossier "resultat" le pdf du rapport, le programme et la documentation Doxygen au format pdf.

## 2 Analyse

### 2.1 Présentation des TAD

#### 2.1.1 TAD FichierTexte

<b>Nom:</b>	FichierTexte
<b>Utilise:</b>	<b>Chaine de caracteres</b> , Mode, <b>Caractere</b> , <b>Booleen</b>
<b>Opérations:</b>	fichierTexte: <b>Chaine de caracteres</b> → FichierTexte ouvrir: FichierTexte × Mode → FichierTexte fermer: FichierTexte → FichierTexte estOuvert: FichierTexte → <b>Booleen</b> Mode: FichierTexte → Mode finFichier: FichierTexte → <b>Booleen</b> ecrireChaine: FichierTexte × <b>Chaine de caracteres</b> → FichierTexte lireChaine: FichierTexte → FichierTexte × <b>Chaine de caracteres</b> ecrireCaractere: FichierTexte × <b>Caractere</b> → FichierTexte lireCaractere: FichierTexte → FichierTexte × <b>Caractere</b>
<b>Sémantiques:</b>	FichierTexte: creation d'un fichier texte à partir d'un fichier identifié par son nom ouvrir: ouvre un fichier texte en lecteur ou ecriture. Si le mode est ecriture et que le fichier existe, alors ce dernier est écrasé fermer: ferme un fichier texte lireCaractere: lit un caractère à partir de la position courante du fichier lireChaine: lit une chaine (jusqu'à un retour à la ligne ou la fin de fichier) à partir de la position courante du fichier ecrireCaractere: écrit un caractère à partir de la position courante du fichier ecrireChaine: écrit une chaine suivie d'un retour à la ligne à partir de la position courante du fichier
<b>Préconditions:</b>	ouvrir(f): non estOuvert(f) fermer(f): estOuvert(f) finFichier(f): mode(f)=lecture lireXX: estOuvert(f) et mode(f)=lecture et non finFichier(f) ecrireXX: estOuvert(f) et mode(f)=ecriture

#### 2.1.2 TAD Mot

<b>Nom:</b>	mot
<b>Utilise:</b>	<b>Chaine de caracteres</b> , NaturelNonNul, <b>Caractere</b> , <b>Booleen</b>
<b>Opérations:</b>	estUneLettre: <b>Caractere</b> → <b>Booleen</b> estUnMot: <b>Chaine de caracteres</b> → <b>Booleen</b>



creerMot: **Chaine de caracteres**  $\rightarrow$  Mot  
 estEgal: Mot  $\times$  Mot  $\rightarrow$  **Booleen**  
 longueur: Mot  $\rightarrow$  NaturelNonNul  
 motEnChaine: Mot  $\rightarrow$  **Chaine de caracteres**  
 remplacerLettre: NaturelNonNul  $\times$  **Caractere**  $\times$  Mot  $\rightarrow$  Mot  
 supprimerLettre: NaturelNonNul  $\times$  Mot  $\rightarrow$  Mot  
 insererLettre: **Caractere**  $\times$  Mot  $\times$  NaturelNonNul  $\rightarrow$  Mot  
 inverserLettre: NaturelNonNul  $\times$  Mot  $\rightarrow$  Mot  
 decomposerMot: Mot  $\times$  NaturelNonNul  $\rightarrow$  Mot  $\times$  Mot

**Sémantiques:** estUneLettre: renvoie vrai si l'élément pris en entrée est une lettre (minuscule ou majuscule)

estUnMot: vérifie que c'est un mot  
 creerUnMot: Créé un mot à partir d'une chaîne de caractères.  
 estEgal: retourne vrai si les mots sont identiques (non sensible à la casse)  
 longueur: calcule la longueur d'un mot  
 motEnChaine: transforme un mot en chaîne de caractères  
 remplacerLettre: remplace une lettre à un certain indice du mot  
 supprimerLettre: supprime une lettre à un certain indice du mot  
 insererLettre: insère une lettre à un certain indice du mot  
 inverserLettre: inverse deux lettres consécutifs du mot  
 decomposerMot: dcompose un mot en deux mots

**Préconditions:** estUnMot(mot): longueur(mot)>1 ET estUneLettre(mot[i])  $\forall 1 \leq i \leq$  longueur(mot)  
 creerMot(c): estUnMot(c)  
 remplacerLettre(i, car, mot):  $i \leq$  longueur(mot)  
 supprimerLettre(i, mot):  $i \leq$  longueur(mot)  
 insererLettre(c, mot, i):  $i < \text{longueur}(\text{mot})$   
 inverserLettre(i, mot): estOuvert(f) et mode(f)=lecture et non finFichier(f)  
 decomposerMot(mot, i):  $1 < i < \text{longueur}(\text{mot})$

### 2.1.3 TAD Dictionnaire

**Nom:** Dictionnaire  
**Utilise:** fichierTexte, Dictionnaire, Mot, **Booleen**, mode  
**Opérations:** dictionnaire:  $\rightarrow$  Dictionnaire  
 estVide: Dictionnaire  $\rightarrow$  **Booleen**  
 ajouterMot: Dictionnaire  $\times$  Mot  $\rightarrow$  Dictionnaire  
 ajouterFichier: Dictionnaire  $\times$  FichierTexte  $\rightarrow$  Dictionnaire  
 chargerDictionnaire: **Chaine de caracteres**  $\rightarrow$  Dictionnaire

sauvegarderDictionnaire: Dictionnaire  $\rightarrow$  **Booleen**

estPresent: Dictionnaire  $\times$  Mot  $\rightarrow$  **Booleen**

**Sémantiques:** dictionnaire: Créé un dictionnaire vide

estVide: vérifie si un dictionnaire est vide

ajouterMot: ajoute un mot dans le dictionnaire

ajouterFichier: ajoute les mots d'un fichier texte dans le dictionnaire

chargerDictionnaire: charge un dictionnaire à partir d'un fichier

sauvegarderDictionnaire: crée un fichier texte contenant tous les mots du dictionnaire mis en paramètre.

estPresent: indique si un mot est présent dans le dictionnaire

**Préconditions:** ajouterFichier(f): mode(fichierTexte) = lecture

ajouterMot(m): non(estPresent(m))

supprimerMot(m): estPresent(m)

#### 2.1.4 TAD CorrecteurOrthographique

**Nom:** CorrecteurOrthographique

**Utilise:** Mot, Dictionnaire, **Booleen**, NaturelNonNul

**Opérations:**

chainesEnMots:	<b>Chaine de caracteres</b> $\rightarrow$ Liste <Mot> $\times$ Liste <Naturel>
sontPresentes:	Liste <Mot> $\times$ Dictionnaire $\rightarrow$ Liste < <b>Booleen</b> >
proposerMots:	Mot $\times$ Dictionnaire $\rightarrow$ Ensemble <Mot>
proposerMotsListe:	Liste <Mot> $\times$ Dictionnaire $\rightarrow$ Liste <Ensemble <Mot> $\times$ Liste < <b>Booleen</b> >

**Sémantiques:**

chainesEnMots:	transforme une chaîne de caractères en une liste de Mot et renvoie aussi une liste de la position de chaque mot
sontPresentes:	renvoie une liste de Booleen indiquant la présence ou non des mots de la liste
proposerMots:	donne un ensemble de Mot correspondant aux corrections possibles d'un mot mal orthographié
proposerMotsListe:	envoie la liste des ensembles de Mot corrigés possible pour les mots qui ne sont pas dans le dictionnaire

## 11



### 3 Conception Préliminaire

#### 3.1 Signature FichierTexte

**fonction** fichierTexte (chaine : **Chaine de caracteres**) : FichierTexte

**procédure** ouvrir (**E/S** fichier : FichierTexte, **E** mode : Mode)

  |précondition(s) non (estOuvert(f))

**procédure** fermer (**E/S** fichier ; FichierTexte)

  |précondition(s) estOuvert(f)

**fonction** estOuvert (fichier : fichierTexte) : **Booleen**

**fonction** Mode (fichier : FichierTexte) : Mode

**fonction** finFichier (fichier : FichierTexte) : **Booleen** mode(fichier)=lecture

**procédure** ecrireChaine (**E/S** fichier : FichierTexte, **E** **Chaine de caracteres**)

  |précondition(s) estOuvert(fichier)  
                  mode(fichier)=écriture

**procédure** lireChaine (**E/S** fichier : FichierTexte, **S** **Chaine de caracteres**)

  |précondition(s) estOuvert(fichier)  
                  mode(fichier)=lecture

**procédure** ecrireCaractere (**E/S** fichier : FichierTexte, **E** **Caractere**)

  |précondition(s) estOuvert(fichier)  
                  mode(fichier)=écriture

**procédure** lireChaine (**E/S** fichier : FichierTexte, **S** **Caractere**)

  |précondition(s) estOuvert(fichier)  
                  mode(fichier)=lecture

#### 3.2 Signature Mot

**fonction** estUneLettre (c : **Caractere**) : **Booleen**

**fonction** longueur (m : **Mot**) : **NaturelNonNul**

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

  |précondition(s) longueur(c)>1 ET estUneLettre(c[i])  $\forall 1 \leq i \leq longueur(m)$

**fonction** creerMot (chaine : **Chaine de caracteres**) : m : **Mot**

  |précondition(s) estUnMot(chaine)

**fonction** sontEgaux (m1, m2 : **Mot**) : **Booleen**

**fonction** motEnChaine (m : **Mot**) : **Chaine de caracteres**

**fonction** remplacerLettre (m : **Mot**, pos : **NaturelNonNul**, c : **Caractere**) : **Mot**

  |précondition(s) pos  $\leq longueur(m)$

**fonction** supprimerLettre (m : **Mot**, pos : **NaturelNonNul**) : **Mot**

$\lfloor \text{précondition(s)} \quad pos \leq \text{longueur}(m)$

**fonction** insererLettre ( $m : \text{Mot}, c : \text{Caractere}, pos : \text{NaturelNonNul}$ ) : Mot

$\lfloor \text{précondition(s)} \quad pos \leq \text{longueur}(m)$

**fonction** inverserLettre ( $m : \text{Mot}, pos : \text{NaturelNonNul}$ ) : Mot

$\lfloor \text{précondition(s)} \quad pos < \text{longueur}(m)$

**fonction** decomposerMot ( $m : \text{Mot}, pos : \text{NaturelNonNul}$ ) : Mot, Mot

$\lfloor \text{précondition(s)} \quad 1 < pos < \text{longueur}(m)$

### 3.3 Signature Dictionnaire

**fonction** Dictionnaire () : Dictionnaire

**fonction** estVide (dictionnaire : Dictionnaire) : **Booleen**

**fonction** estPrésent (dictionnaire : Dictionnaire, motChercher : mot) : **Booleen**

**procédure** ajouterMot (**E/S** dictionnaire : Dictionnaire, **E** mot : Mot)

$\lfloor \text{précondition(s)} \quad \text{non}(\text{estPrésent}(\text{mot}))$

**procédure** ajouterFichier (**E/S** dictionnaire : Dictionnaire, **E** fichier : FichierTexte)

$\lfloor \text{précondition(s)} \quad \text{mode}(\text{fichierTexte}) = \text{lecture}$

**procédure** chargerDictionnaire (**E** chaine : **Chaine de caracteres**, **S** dictionnaire : Dictionnaire)

**fonction** enregistrerDictionnaire (dictionnaire : Dictionnaire) : FichierTexte

### 3.4 Signature CorrecteurOrthographique

**fonction** chaineEnMots ( $c : \text{Caractere}$ ) : Liste<Mot>, Liste<Naturel>

**fonction** sontPresentes (mots : Liste<Mot>, d : Dictionnaire) : Liste<Booleen>

**fonction** proposerMot ( $m : \text{Mot}, d : \text{Dictionnaire}$ ) : Ensemble<Mot>

**fonction** proposerMotListe (liste : Liste<Mot>, d : Dictionnaire) : Liste<Ensemble<Mot>, Liste<Booleen>

## 4 Conception Détaillé

### 4.1 Conception Détaillé du TAD Fichier Texte

### 4.2 Conception Détaillé du TAD mot

**fonction** estUneLettre (c : Caractere) : Booleen

**Déclaration** res : Booleen

**debut**

**si** ( $'a' \geq c \leq 'z'$ )  $\vee$  ( $'A' \geq c \leq 'Z'$ ) **alors**

res  $\leftarrow$  vrai

**sinon**

res  $\leftarrow$  faux

**finsi**

**retourner** res

**fin**

/\*\*\*\*\*

**fonction** longueur (mot : Mot) : Entier

**Déclaration** long : Entier

**debut**

long  $\leftarrow$  0

**tant que** mot[long]  $\neq$  '0' **faire**

long  $\leftarrow$  long+1

**fintantque**

**retourner** long

**fin**

/\*\*\*\*\*

**fonction** estUnMot (m : Chaîne de caracteres) : Booleen

**[précondition(s)]** (longueur(m)>1) ET (estUneLettre(m[i]))

**Déclaration** i : Entier, res : Booleen, c : Caractere

**debut**

**si** longueur(m)  $\leq$  1 **alors**

**retourner** vrai

**sinon**

**pour** i  $\leftarrow$  1 à longueur(m) **faire**

c  $\leftarrow$  m[i]

**si** estUneLettre(c) **alors**

res  $\leftarrow$  vrai

**sinon**

res  $\leftarrow$  faux

**finsi**

```

    finpour
    fin
    retourner res
fin
/*****/
fonction créerMot (chaîne : Chaîne de caracteres) : m : Mot
    Déclaration  i : Entier, longueur : Entier, lettre : Caractere
debut
    i ← 1
    longueur ← 0
    lire(lettre)
    tant que estUneLettre(lettre) faire
        m[i] ← lettre
        i ← i+1
        longueur ← longueur+1
    fintantque
    retourner m
fin
/*****/
fonction sontEgaux (mot1,mot2 : Mot) : Booleen
    Déclaration  res : Booleen, i : Entier
debut
    si longueur(mot1) ≠ longueur(mot2) alors
        res ← faux
    sinon
        res ← vrai
        i ← 0
        tant que (res = vrai) ∧ & (i ≤ longueur(mot1)) faire
            si mot1[i] = mot2[i] alors
                res ← vrai
                i ← i+1
            sinon
                res ← faux
        finsi
    fintantque
    finsi
    retourner Res
fin
/*****/
fonction SupprimerLettre (pos : Entier, mot : Mot) : Mot
    précondition(s)  pos ≤ longueur(mot)

```

```

debut
  tant que mot[pos] ≠ '0' faire
    mot[pos] ← mot[pos+1]
    pos ← pos+1
  fintantque
fin
retourner mot
/*****/
fonction InsérerLettre (pos : Entier, c : Caractere, mot : Mot) : Mot
  [précondition(s) (pos ≤ longueur(mot)+1)) ∧ & (estUneLettre(c))
debut
  mot[pos] ← c
  tant que mot[pos] ≠ '0' faire
    mot[pos+1] ← mot[pos]
    pos ← pos+1
  fintantque
  retourner mot
fin
/*****/
fonction RemplacerLettre (position : Entier, c : Caractere, mot : Mot) : Mot
debut
  supprimerLettre(position, mot)
  insérerLettre(position, c, mot)
  retourner mot
fin
/*****/
fonction inverserDeuxLettresConsecutives (position : Entier) : mot : Mot
  [précondition(s) pos < longueur(mot)]
  Déclaration a : Caractere
debut
  a ← mot[pos]
  mot[pos] ← mot[pos+1]
  mot[pos+1] ← a
fin
/*****/
fonction décomposerMot (pos : Entier, unMot : Mot) : Mot, Mot
  [précondition(s) (pos < longueur(unMot)) ∧ & (pos > 1)]
  Déclaration i : Entier, mot1, mot2 : Mot
debut

```



```

pour i ← 1 à pos faire
    mot1[i] ← unMot[i]
finpour
pour i ← pos+1 à longueur(unMot) faire
    mot2[i] ← unMot[i]
finpour
retourner mot1
retourner mot2
fin

```

### 4.3 Conception Détaillé du TAD Dictionnaire

**Type** ArbreBinaire =  $\wedge$ Noeud

**Type** Noeud = **Structure**

```

mot : Mot
filsG : ArbreBinaire
filsD : ArbreBinaire

```

**finstructure**

**Type** Dictionnaire = ArbreBinaire<Mot>

**fonction** dictionnaire (m : Mot) : Dictionnaire

**Déclaration** resultat : Dictionnaire

**debut**

```

resultat.mot ← m
resultat.filsG ← null
resultat.filsD ← null
retourner resultat

```

**fin**

**fonction** estPresent (d : Dictionnaire, m : Mot) : **Booleen**

**debut**

```

si estVide(d) alors
    retourner estPresent(obtenirFilsGauche(d),m)
sinon
    retourner estPresent(obtenirFilsDroite(d),m)
finsi

```

**fin**

**procédure** faireSimpleRotationDroite (**E/S** d : Dictionnaire)

**[précondition(s)]** non(estVide(obtenirFilsGauche(d)))

**Déclaration** fg, fd, fgg, fdg : Dictionnaire

**debut**

```
fg ← obtenirFilsGauche(d)
fd ← obtenirFilsDroit(d)
fgg ← obtenirFilsGauche(fg)
fdg ← obtenirFilsDroit(fg)
d ← fixerRacine(fgg, fixerRacine(fdg, fd, obtenirMot(d)), obtenirMot(fg))
```

**fin**

**procédure** faireSimpleRotationGauche (**E/S** d : Dictionnaire)

  |**précondition**(s) non(estVide(obtenirFilsDroit(d)))

**Déclaration** fg, fd, fgd, fdd : Dictionnaire

**debut**

```
fg ← obtenirFilsGauche(d)
fd ← obtenirFilsDroit(d)
fgd ← obtenirFilsGauche(fd)
fdd ← obtenirFilsDroit(fd)
d ← fixerRacine(fixerRacine(fg, fgd, obtenirElement(d)), fdd, obtenirElement(fd))
```

**fin**

**procédure** faireDoubleRotationDroite (**E/S** d : Dictionnaire)

  |**précondition**(s) non(estVide(obtenirFilsGauche(d))) ET  
                    non(estVide(obtenirFilsDroit(obtenirFilsGauche(d))))

**Déclaration** fg : Dictionnaire

**debut**

```
fg ← obtenirFilsGauche(d)
faireSimpleRotationGauche(fg)
fixerFilsGauche(d, fg)
faireSimpleRotationDroite(d)
```

**fin**

**procédure** faireDoubleRotationGauche (**E/S** d : Dictionnaire)

  |**précondition**(s) non(estVide(obtenirFilsDroit(d))) ET  
                    non(estVide(obtenirFilsGauche(obtenirFilsDroit(d))))

**Déclaration** fd : Dictionnaire

**debut**

```
fd ← obtenirFilsDroit(d)
faireSimpleRotationDroite(fd)
fixerFilsDroit(d, fd)
faireSimpleRotationGauche(d)
```

```

fin
procédure ajouterMot (E/S d : Dictionnaire, E m : Mot)
  |précondition(s)  non(estPresent(d,m))
  Déclaration  temp : Dictionnaire
debut
  si estPlusPetit(m, obtenirMot(d)) alors
    temp ← obtenirFilsGauche(d)
    inserer(temp, m)
    fixerFilsGauche(d, temp)
  si hauteur(obtenirFilsGauche(d)) > hauteur(obtenirFilsDroit(d))+1 alors
    si hauteur(hauteur(obtenirFilsGauche(obtenirFilsGauche(d))) ≥
      hauteur(obtenirFilsDroit(obtenirFilsGauche(d))) alors
      faireSimpleRotationDroite(d)
    sinon
      faireDoubleRotationDroite(d)
    finsi
  finsi
sinon
  temp ← obtenirFilsDroit(d)
  inserer(temp, m)
  fixerFilsDroit(d, temp)
  si hauteur(obtenirFilsDroit(d)) > hauteur(obtenirFilsGauche(d))+1 alors
    si hauteur(obtenirFilsGauche(obtenirFilsDroit(d))) ≤
      hauteur(obtenirFilsDroit(obtenirFilsDroit(d))) alors
      faireSimpleRotationGauche(d)
    sinon
      faireDoubleRotaionGauche(d)
    finsi
  finsi
fin
procédure enregistrerDicoR (E/S f : fichierTexte, E d : Dictionnaire)
  |précondition(s)  mode(f) = ecriture
  Déclaration  fg, fd : Dictionnaire
debut
  si estVide(d) alors
    f ← ecrireChaine(f, "")
  sinon
    f ← ecrireChaine(f, motEnChaine(obtenirElement(d)))

```

```

    fg ← obtenirFilsGauche(d)
    enregistrerDicoR(f, fg)

    fd ← obtenirFilsDroit(d)
    enregistrerDicoR(f, fd)
  fin
fin
procédure enregistrerDico (E d : Dictionnaire, nomFichier : ChaîneDeCaractere)
  |précondition(s)  nomFichier != ""
  Déclaration  f : fichierTexte
debut
  f ← fichierTexte(nomFichier)
  f ← ouvrir(f, ecriture)
  enregistrerDicoR(f, d)
fin
procédure chargerDicoR (E/S d : Dictionnaire, f : FichierTexte)
  |précondition(s)  mode(f) = lecture
  Déclaration  temp, fd, fg : Dictionnaire
                 nvChaine : Chaîne
debut
  f, nbChaine ← lireChaine(f)

  si Chaîne.longueur(nvChaine) != 0 alors
    d ← ajouterRacine(arbreBinaire(), arbreBinaire(), creerMot(chaine))

    si non(finFichier(f)) alors
      fg ← obtenirFilsGauche(d)
      chargerDicoR(fg, f)
      fixerFilsGauche(d, fg)
    fin

    si non(finFichier(f)) alors
      fd ← obtenirFilsDroit(d)
      chargerDicoR(fd, f)
      fixerFilsDroit(d, fd)
    fin
  fin
fin
fonction chargerDico (nomFichier : ChaîneDeCaractere) : Dictionnaire

```

```
[précondition(s)  nomFichier != ""
```

```
Déclaration  resultat : Dictionnaire
               f : FichierTexte
               chaine : ChaineDeCaractere
```

```
debut
```

```
  resultat ← dictionnaire()
  f ← fichierTexte(nomFichier)
  f ← ouvrir(f, lecture)
```

```
  si non(finFichier(f)) alors
    f, chaine ← lireChaine(f)
    chargerDicoR(resultat, f, chaine)
```

```
  finsi
```

```
  f ← fermer(f)
```

```
  retourner resultat
```

```
fin
```

#### 4.4 Conception Détaillé du TAD Correcteur Orthographique

```
Constante alphabet = 'abcdefghijklmnopqrstuvwxyzèèèèââââûûûûôôôôÿç'
```

```
fonction sontPresent (mots : Liste<Mot>, d : Dictionnaire) : Liste<Booleen>
```

```
  Déclaration  i : Naturel, resultat : Liste<Booleen>
```

```
debut
```

```
  pour i ← 1 à Liste.longueur(mots) faire
    ajouter(resultat, estPresent(Liste.obtenirElement(mots, i), d))
```

```
  finpour
```

```
  retourner resultat
```

```
fin
```

```
fonction proposerMots (m : Mot, d : Dictionnaire) : Ensemble<Mot>
```

```
  Déclaration  motCorrige, motCorrige2 : Mot, resultat : Ensemble<Mot>
```

```
debut
```

```
  pour k ← 1 à Chaine.longueur(alphabet) faire
    pour i ← 1 à Mot.longueur(m) faire
      motCorrige <- remplacerLettre(m, alphabet[k], i)
```

```
    si Dictionnaire.estPresent(motCorrige, d) alors
      ajouter(resultat, motCorrige)
```

```
    finsi
```

```

finpour
pour i ← 1 à Mot.longueur(m) + 1 faire
    motCorrige <- insererLettre(m, alphabet[k], i)

    si Dictionnaire.estPresent(motCorrige, d) alors
        ajouter(resultat, motCorrige)
    finsi
finpour
finpour
pour i ← 1 à Mot.longueur(m) faire
    motCorrige <- supprimerLettre(m, i)

    si Dictionnaire.estPresent(motCorrige, d) alors
        ajouter(resultat, motCorrige)
    finsi
finpour
pour i ← 2 à Mot.longueur(m) - 1 faire
    motCorrige, motCorrige2 <- decomposerMot(m, i)

    si Dictionnaire.estPresent(motCorrige, d) alors
        ajouter(resultat, motCorrige)
    finsi
    si Dictionnaire.estPresent(motCorrige2, d) alors
        ajouter(resultat, motCorrige2)
    finsi
finpour
retourner resultat
fin

fonction proposerMotsListe (mots : Liste<Mot>, d : Dictionnaire) : Liste< Ensemble<Mot> >
    Déclaration resultat : Liste< Ensemble<Mot> >
debut
    pour i ← 1 à Liste.longueur(mots) faire
        ajouter(resultat, proposerMots(obtenirElements(mots, i), d))
    finpour
fin

```

## 5 Implémentation de la conception en C

### 5.1 TAD mot

#### 5.1.1 Fichier mot.h

---

#### 5.1.2 Fichier mot.c

---

#### 5.1.3 Fichier testsMot.c

---

### 5.2 TAD dictionnaire

#### 5.2.1 Fichier dictionnaire.h

---

#### 5.2.2 Fichier dictionnaire.c

---

#### 5.2.3 Fichier testsDictionnaire.c

---

### 5.3 TAD correcteurOrthographique

#### 5.3.1 Fichier correcteurOrthographique.h

---

#### 5.3.2 Fichier correcteurOrthographique.c

---

#### 5.3.3 Fichier testsCorrecteurOrthographique.c

---

## 6 Conclusion

### 6.1 Conclusions personnelles

#### 6.1.1 Conclusion Alexis IMBERT

#### 6.1.2 Conclusion Ruth LIOTÉ

#### 6.1.3 Conclusion Amina SAKOUTI

#### 6.1.4 Conclusion Léo ZEDEK

### 6.2 conclusion générale du projet