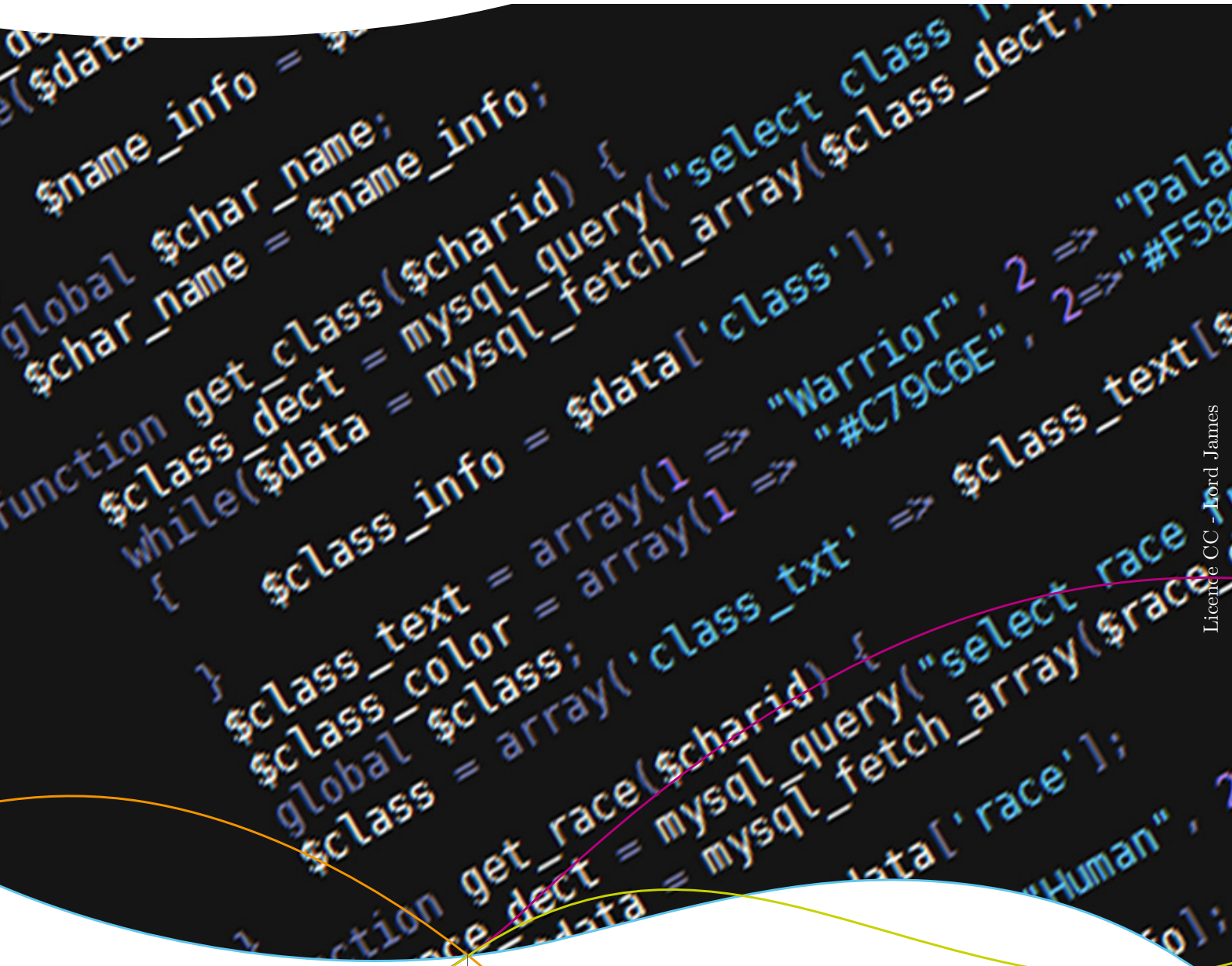
INSA ROUEN
À taille humaine,
à l'échelle du monde





Projet d'Algo

ASI3 2013-2014

# Correcteur orthographique

## *Rapport de projet*

Antoine Augusti
Etienne Batise
Jean-Claude Bernard
Thibaud Dauce
Faustine Demiselle

# Table des matières

# Introduction

À la question *« Pourquoi réaliser un correcteur orthographique en C ? »* nous répondons tous en coeur : « parce que on nous l'a demandé ! ». D'après Wikipedia, voici la définition d'un correcteur orthographique :

> « Un correcteur orthographique est, en informatique, un outil logiciel permettant d'analyser un texte afin de détecter, et éventuellement de corriger, les fautes d'orthographe et les coquilles qu'il contient. »

Le but de ce projet d'algorithmique est de réaliser un correcteur orthographique, capable de détecter des erreurs de langue à l'aide d'un dictionnaire et de proposer des corrections orthographiques pertinentes, puisées dans le dictionnaire donné.

Aujourd'hui tout le monde utilise au quotidien un correcteur orthographique, celui-ci étant le plus souvent déjà présent dans les smartphones et ordinateurs. Tout le monde s'est déjà fait sauver par un programme qui corrige une coquille de frappe lors de l'envoi d'un SMS ou d'un e-mail important. Un logiciel de correction orthographique ne peut pas remplacer un académicien chevronné mais permet de corriger une bonne partie des fautes d'orthographe que nous commettons, plus ou moins involontairement.

Pour nous aider dans la réalisation de ce projet, nous avons à notre disposition un dictionnaire de la langue française contenant plus de 300 000 mots. Ce dictionnaire doit pouvoir nous aider à proposer des corrections pertinentes pour une phrase donnée (contenant potentiellement des fautes) en argument à notre programme. Notre programme a pour objectif d'être proche du programme *Aspell* du projet GNU.

Nous commencerons tout d'abord ce rapport par la définition des TAD utilisés ainsi que par l'analyse descendante correspondante au projet. Nous enchaînerons par la suite par la conception préliminaire puis détaillée du problème donné. Enfin nous terminerons par l'ensemble du code C de notre projet ainsi que les différents tests unitaires effectués.

# Première partie

# Analyse

# Chapitre 1

# TAD

## Sommaire



## 1.1 TAD Mot

**Nom** : Mot

**Utilise** : **Booléen**,**Naturel**, **NaturelNonNul**, **Caractère**, **Chaîne de caractères**

**Opérations** :
mot : $\rightarrow$ **Mot**
obtenirContenu : **Mot** $\rightarrow$ **Chaîne de caractères**
sontEgaux : **Mot** $\times$ **Mot** $\rightarrow$ **Booléen**
longueur : **Mot** $\rightarrow$ **Naturel**
insererCaractere : **Mot** $\times$ **Caractère** $\times$ **NaturelNonNul** $\nrightarrow$ **Mot**
modifierCaractere : **Mot** $\times$ **Caractère** $\times$ **NaturelNonNul** $\nrightarrow$ **Mot**
supprimerCaractere : **Mot** $\times$ **NaturelNonNul** $\nrightarrow$ **Mot**
iemeCaractere : **Mot** $\times$ **NaturelNonNul** $\nrightarrow$ **Caractère**
concatener : **Mot**dDeuxParams**MotMot** $\rightarrow$ **Mot**
estUnMotValide : **Mot**t $\rightarrow$ **Booléen**
separerMots : **Mot** $\times$ **Naturel** $\nrightarrow$ **Mot** $\times$ **Mot**
sousMot : **Mot** $\times$ **Naturel** $\times$ **Naturel** $\nrightarrow$ **Mot**
estSousMot : **Mot** $\times$ **Mot** $\rightarrow$ **Booléen**
sontIdentiques : **Mot** $\times$ **Mot** $\rightarrow$ **Booléen**

**Sémantiques** :
sontEgaux : tous les caract ?res sont ?gaux (par valeurs).
estUnMotValide : le mot ne contient que des lettres ou des tirets.
sontIdentiques : les mots sont ?gaux (par adresse).
separerMots : S ?pare en deux mots. L'indice donn ? est le d ?but du deuxi ?me mot. separerMots(abc, 2) donne a et bc.
estSousMot : d ?termine si un mot est contenu dans un autre mot, n'importe o ? dans ce dernier.

**Axiomes** :
- longueur(insererCaractere(mot, c, i)) = longueur(mot) + 1
- longueur(modifierCaractere(mot, c, i)) = longueur(mot)
- longueur(supprimerCaractere(mot, c, i)) = longueur(mot) - 1
- sontEgaux(supprimerCaractere(insererCaractere(mot, c, i), i), mot)
- sontEgaux(insererCaractere(supprimerCaractere(mot, i), c, i), mot)
- iemeCaractere(insererCaractere(mot, c, i), i) = c
- longueur(concatener(mot1, mot2)) = longueur(mot1) + longueur(mot2)
- iemeCaractere(concatener(mot1, mot2), longueur(mot1) + 1) = iemeCaractere(mot2, 1)
- separerMots(concatener(mot1, mot2), longueur(mot1) + 1) = mot1 , mot2

**Préconditions** : insererCaractere : $1 \leq i \leq longueur(mot) + 1$ et estLettreOuTiret(c)
modifierCaractere : $1 \leq i \leq longueur(mot)$ et estLettreOuTiret(c)
supprimerCaractere : $1 \leq i \leq longueur(mot)$
separerMots : $1 \leq i \leq longueur(mot)$
iemeCaractere : $1 \leq i \leq longueur(mot)$ et non(longueur(mot) = 0)
sousMot : $1 \leq debut + longueurSousMot \leq longueur(mot)$ et non(longueur(mot) = 0)

## 1.2 TAD SuperMot

**Nom** : SuperMot
**Utilise** : **Booléen**, **NaturelNonNul**, **Mot**, **Liste<Mot>**
**Opérations** : superMot : $\rightarrow$ **SuperMot**
obtenirMot : **SuperMot** $\rightarrow$ **Mot**
obtenirPosition : **SuperMot** $\rightarrow$ **NaturelNonNul**
obtenirListeDeCorrection : **SuperMot** $\rightarrow$ **Liste<Mot>**
obtenirValidite : **SuperMot** $\rightarrow$ **Booléen**
fixerMot : **SuperMot** $\times$ **Mot** $\rightarrow$ **SuperMot**
fixerPosition : **SuperMot** $\times$ **NaturelNonNul** $\rightarrow$ **SuperMot**
fixerListeDeCorrection : **SuperMot** $\times$ **Liste<Mot>** $\rightarrow$ **SuperMot**
fixerValidite : **SuperMot** $\times$ **Booléen** $\rightarrow$ **SuperMot**
**Sémantiques** : obtenirPosition : La position du mot dans la phrase en entrée.
obtenirListeDeCorrection : La liste des corrections pour le mot.
obtenirValidite : Indique si le mot est bien orthographié ou pas.

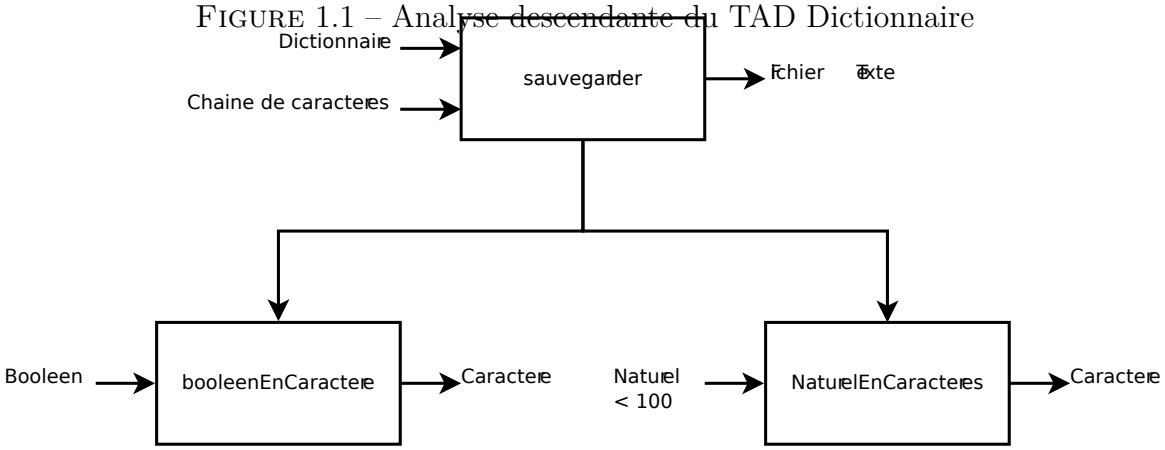
## 1.3 TAD Dictionnaire

**Nom** : Dictionnaire
**Utilise** : **Booléen**, **Naturel**, **Mot**, **FichierTexte**, **Chaîne de caractères**, **ElementDictionnaire**
**Opérations** : dictionnaire : $\rightarrow$ **Dictionnaire**
estVide : **Dictionnaire** $\rightarrow$ **Booléen**
insererMot : **Dictionnaire** $\times$ **Mot** $\rightarrow$ **Dictionnaire**
insererFichier : **Dictionnaire** $\times$ **Chaîne de caractères** $\rightarrow$ **Dictionnaire**
estPresent : **Dictionnaire** $\times$ **Mot** $\rightarrow$ **Booléen**
charger : **Chaîne de caractères** $\rightarrow$ **Dictionnaire**
sauvegarder : **Dictionnaire** $\times$ **Chaîne de caractères** $\rightarrow$ **FichierTexte**

**Axiomes** :

- estVide(dictionnaire())
- non estVide(insererMot(d, m))
- estPresent(insererMot(d, m), m)
- non(estPresent(d, m) XOR insererMot(d, m) = d)
- non estPresent(d, m) ET nombreMots(insererMot(d, m)) = 1 + nombreMots(d)

FIGURE 1.1 – Analyse descendante du TAD Dictionnaire



## 1.4 TAD ElementDictionnaire

**Nom** : ElementDictionnaire

**Utilise** : **Booléen**, **Caractère**, **Naturel**, **NaturelNonNul**

**Opérations** :
element : **Caractère** $\times$ **Booléen** $\rightarrow$ **ElementDictionnaire**
ajouterElementSuivant : **ElementDictionnaire** $\times$ **ElementDictionnaire** $\rightarrow$ **ElementDictionnaire**
supprimerElementSuivant : **ElementDictionnaire** $\times$ **NaturelNonNul** $\nrightarrow$ **ElementDictionnaire**
obtenirTableauDesElementsSuivants : **ElementDictionnaire** $\rightarrow$ **Tableau**[1...NBLETTRES] **de** $\hat{}$**ElementDictionnaire**
obtenirLongueurTableauDesElementsSuivants : **ElementDictionnaire** $\rightarrow$ **Naturel**
obtenirLettre : **ElementDictionnaire** $\rightarrow$ **Caractère**
obtenirBooleen : **ElementDictionnaire** $\rightarrow$ **Booléen**

**Préconditions** : supprimerElementSuivant(e, i) : $1 \leq i \leq$ obtenirLongueurTableauDesElementsSuivants(e)

## 1.5 TAD Parametres

**Nom** : Parametres

**Utilise** : **Booléen**, **Naturel**, **Chaîne de caractères**

**Opérations** :
parametres : $\rightarrow$ **Parametres**
fixerDictionnaire : **Parametres** $\times$ **Chaîne de caractères** $\rightarrow$ **Parametres**
fixerFichier : **Parametres** $\times$ **Chaîne de caractères** $\rightarrow$ **Parametres**
fixerCible : **Parametres** $\times$ **Chaîne de caractères** $\rightarrow$ **Parametres**
fixerAide : **Parametres** $\times$ **Booléen** $\rightarrow$ **Parametres**
obtenirDictionnaire : **Parametres** $\rightarrow$ **Chaîne de caractères**
obtenirFichier : **Parametres** $\rightarrow$ **Chaîne de caractères**
obtenirAide : **Parametres** $\rightarrow$ **Booléen**
obtenirCible : **Parametres** $\rightarrow$ **Chaîne de caractères**

## 1.6 TAD Correcteur Orthographique

**Nom** : CorrecteurOrthographique

**Utilise** : **Booléen**, **Naturel**, **Mot**, **Dictionnaire**, **Liste<Mot>**

**Opérations** : proposerMots : **Dictionnaire** $\times$ **Mot** $\nrightarrow$ **Liste<Mot>**

**Préconditions** : proposerMots(d, m) : non (longueur(m) = 0)

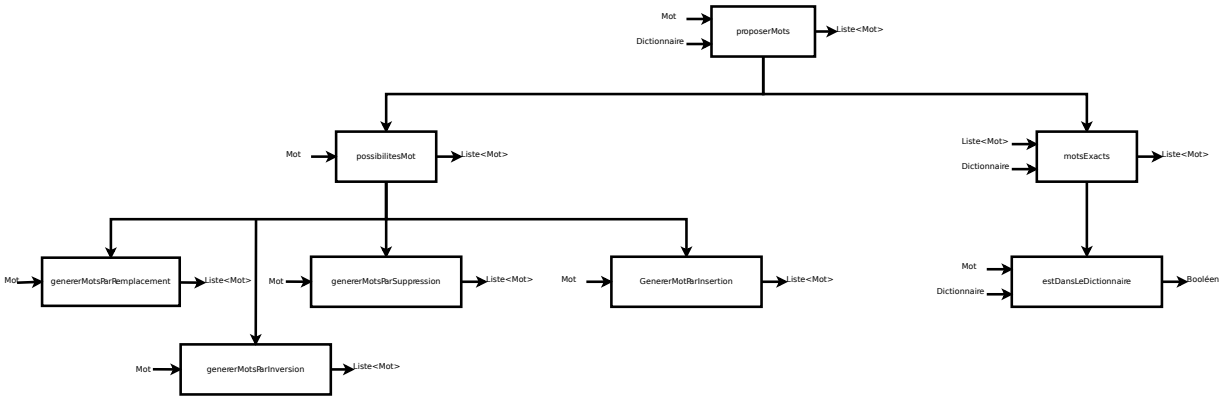



FIGURE 1.2 – Analyse descendante du TAD Correcteur Orthographique

# Chapitre 2

# Analyse Descendante

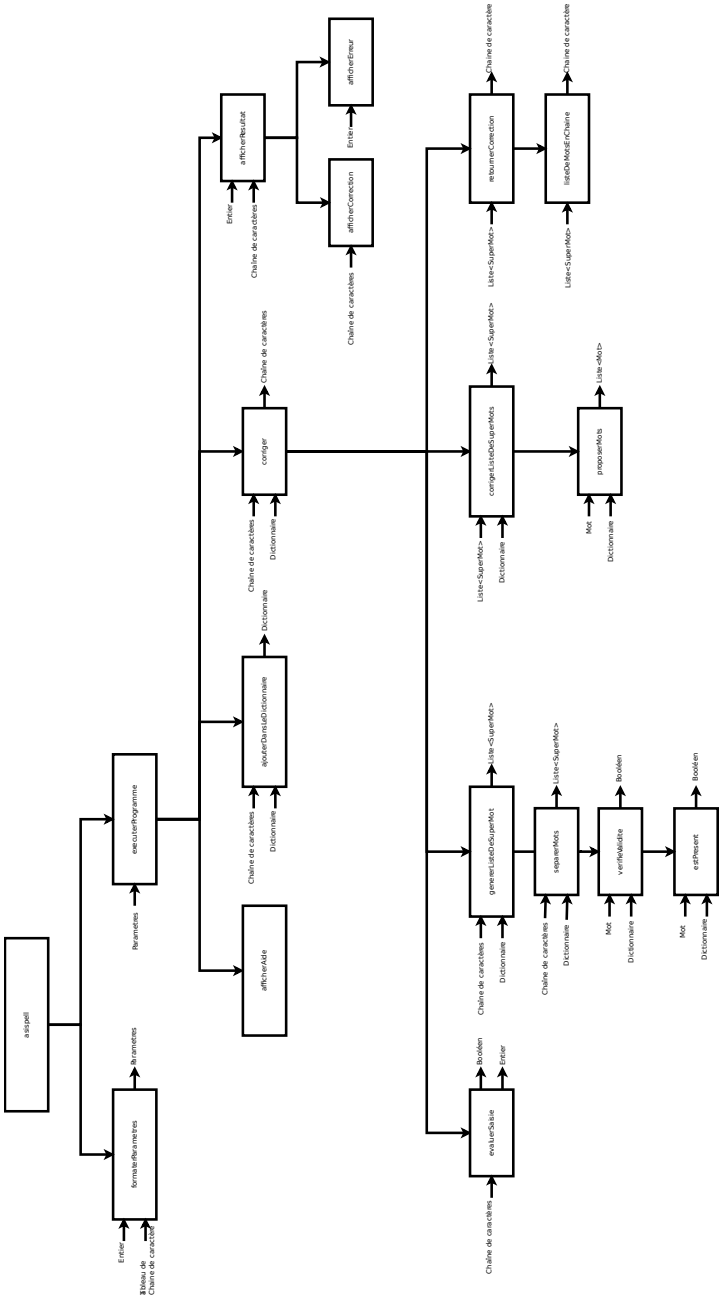



FIGURE 2.1 – Analyse descendante du programme

# Deuxième partie

# Conception préliminaire

# Chapitre 1

# Conception préliminaire des TAD

## Sommaire



## 1.1 TAD Mot

**fonction** mot () **: Mot**
**fonction** obtenirContenu (mot : **Mot**) **: Chaîne de caractères**
**fonction** sontEgaux (mot1, mot2 : **Mot**) **: Booléen**
**fonction** longueur (mot : **Mot**) **: Naturel**
**procédure** inserer Caractere (**E/S** mot : **Mot** ; **E** c : **Caractère**, i : **NaturelNonNul**)
⌊**précondition(s)** $1 \leq i \leq$ longueur(mot) + 1 ET estLettreOuTiret(c)
**procédure** modifierCaractere (**E/S** mot : **Mot** ; **E** c : **Caractère**, i : **NaturelNonNul**)
⌊**précondition(s)** $1 \leq i \leq$ longueur(mot) ET estLettreOuTiret(c)
**procédure** supprimerCaractere (**E/S** mot : **Mot** ; **E** i : **NaturelNonNul**)
⌊**précondition(s)** $1 \leq i \leq$ longueur(mot)
**fonction** iemeCaractere (mot : **Mot**, i : **NaturelNonNul**) **: Caractère**
⌊**précondition(s)** $1 \leq i \leq$ longueur(mot)
**fonction** concatener (a : **Mot**, b : **Mot**) **: Mot**
**fonction** estUnMotValide (mot : **Mot**) **: Booléen**
**procédure** separerMots (**E** mot : **Mot**, i : **NaturelNonNul** ; **S** a : **Mot**, b : **Mot**)
⌊**précondition(s)** $1 \leq i \leq$ longueur(mot)
**fonction** sousMot (mot : **Mot**, debut : **NaturelNonNul**, longueur : **NaturelNonNul**) **: Mot**
⌊**précondition(s)** $1 \leq$ debut + longueur $\leq$ longueur(mot) et non(longueur(mot) = 0
**fonction** estSousMot (a : **Mot**, b : **Mot**) **: Booléen**
**fonction** sontIdentiques (a : **Mot**, b : **Mot**) **: Booléen**

## 1.2 TAD SuperMot

**fonction** superMot () **: SuperMot**
**fonction** obtenirMot (supermot : **SuperMot**) **: Mot**
**procédure** fixerMot (**E/S** supermot : **SuperMot**, **E** mot : **Mot**)
**fonction** obtenirPosition (supermot : **SuperMot**) **: NaturelNonNul**
**procédure** fixerPosition (**E/S** supermot : **SuperMot**, **E** position : **NaturelNonNul**)
**fonction** obtenirListeDeCorrection (supermot : **SuperMot**) **: Liste<Mot>**
**procédure** fixerListeDeCorrection (**E/S** supermot : **SuperMot**, **E** corrections : **Liste<Mot>**)
**fonction** obtenirValidite (supermot : **SuperMot**) **: Booléen**
**procédure** fixerValidite (**E/S** supermot : **SuperMot**, **E** estValide : **Booléen**)

## 1.3 TAD Dictionnaire

**fonction** dictionnaire () **: Dictionnaire**
**fonction** estVide (dictionnaire : **Dictionnaire**) **: Booléen**
**procédure** insererMot (**E/S** dictionnaire : **Dictionnaire** ; **E** mot : **Mot**)
**procédure** insererFichier (**E/S** dictionnaire : **Dictionnaire** ; **E** chaine : **Chaîne de caractères**)
**fonction** estPresent (dictionnaire : **Dictionnaire**, mot : **Mot**) **: Booléen**
**fonction** charger (f : **Chaîne de caractères**) **: Dictionnaire**
**fonction** sauvegarder (dictionnaire : **Dictionnaire**, emplacement : **Chaîne de caractères**) **: Fichier-Texte**

## 1.4 TAD ElementDictionnaire

**fonction** element (c : **Caractère**, b : **Booléen**) **: ElementDictionnaire**
**procédure** ajouterElementSuivant (**E/S** e1 : **ElementDictionnaire** ; **E** e2 : **ElementDictionnaire**)
**procédure** supprimerElementSuivant (**E/S** e : **ElementDictionnaire** ; **E** indice : **NaturelNonNul**)
⌊**précondition(s)** $1 \leq$ indice $\leq$ obtenirLongueurTableauDesElementsSuivants(e)
**fonction** obtenirTableauDesElementsSuivants (**E** e : **ElementDictionnaire**) **: Tableau**[1...NBLETTRES] **de ElementDictionnaire**
**fonction** obtenirLongueurTableauDesElementsSuivants (**E** e : **ElementDictionnaire**) **: Naturel**
**fonction** obtenirLettre (e : **ElementDictionnaire**) **: Caractère**
**fonction** obtenirBooleen (e : **ElementDictionnaire**) **: Booléen**

## 1.5 TAD Correcteur Orthographique

*// Fonctions et procédures publiques*
**fonction** proposerMots (dictionnaire : **Dictionnaire**, mot : **Mot**) **: Liste<Mot>**
⌊**précondition(s)** longueur(mot) $\geq 1$
*// Fonctions et procédures privées*
**fonction** possibilitesMot (mot : **Mot**) **: Liste<Mot>**
⌊**précondition(s)** longueur(mot) $\geq 1$
**fonction** genererMotsParRemplacement (mot : **Mot**) **: Liste<Mot>**
⌊**précondition(s)** longueur(mot) $\geq 1$
**fonction** genererMotsParInversion (mot : **Mot**) **: Liste<Mot>**

⌊**précondition(s)** longueur(mot) ≥ 1
**fonction** genererMotsParSuppression (mot : **Mot**) **: Liste**<**Mot**>
⌊**précondition(s)** longueur(mot) ≥ 1
**fonction** genererMotsParInsertion (mot : **Mot**) **: Liste**<**Mot**>
⌊**précondition(s)** longueur(mot) ≥ 1
**fonction** motsExacts (liste : **Liste**<**Mot**>, d : **Dictionnaire**) **: Liste**<**Mot**>

## 1.6 TAD Parametres

**fonction** parametres () **: Parametres**
**procédure** fixerDictionnaire (**E/S** p : **Parametres**, **E** c : **Chaîne de caractères**)
**procédure** fixerFichier (**E/S** p : **Parametres**, **E** c : **Chaîne de caractères**)
**procédure** fixerCible (**E/S** p : **Parametres**, **E** c : **Chaîne de caractères**)
**procédure** fixerAide (**E/S** p : **Parametres**, **E** b : **Booléen**)
**fonction** obtenirDictionnaire (**Parametres**) **: Chaîne de caractères**
**fonction** obtenirFichier (**Parametres**) **: Chaîne de caractères**
**fonction** obtenirAide (**Parametres**) **: Booléen**
**fonction** obtenirCible (**Parametres**) **: Chaîne de caractères**

# Chapitre 2

# Conception préliminaire de la logique métier

**procédure** asispell (**E** argc : **Naturel**, argv : **Tableau**[1..nbParametres] **de Chaîne de caractères**)
**fonction** formaterParametres (nbParametre : **Entier**, parametres : **Tableau**[1..nbParametres] **de Chaîne de caractères**) **: Parametres**
**procédure** executerProgramme (**E** parametres : **Parametres**)
**procédure** ajouterDansLeDictionnaire ()
**fonction** corriger (chaine : **Chaîne de caractères**, dictionnaire : **Dictionnaire**) **: Chaîne de caractères**
**procédure** evaluerSaisie (**E/S** estValide : **Booléen**, erreur : **Entier** ; **E** chaine : **Chaîne de caractères**)
**fonction** genererListeDeSuperMots (chaine : **Chaîne de caractères**, dictionnaire : **Dictionnaire**) **: Liste<SuperMot>**
**fonction** separerMots (chaine : **Chaîne de caractères**, dictionnaire : **Dictionnaire**) **: Liste<SuperMot>**

**fonction** verifieValidite (mot : **Mot**, dictionnaire **Dictionnaire**) **: Booléen**
**procédure** corrigerListeDeSuperMots (**E/S** listeDeSuperMots : **Liste<SuperMot>** ; **E** dictionnaire : **Dictionnaire**)
**fonction** retournerCorrection (listeDeSuperMots : **Liste<SuperMot>**) **: Chaîne de caractères**
**fonction** listeDeMotsEnChaine (listeDeMots : **Liste<Mot>**) **: Chaîne de caractères**
**procédure** afficherResultat (**E** erreur : **Entier**, chaine : **Chaîne de caractères**)
**procédure** afficherErreur (**E** typerDErreur : **Entier**)
**procédure** afficherCorrection (**E** correction : **Chaîne de caractères**)
**procédure** afficherAide ()

# Troisième partie

# Conception détaillée

# Chapitre 1

# Conception détaillée des TAD

## Sommaire



## 1.1 TAD Mot

**Type** Mot = **Structure**
  contenu : **Chaîne de caractères**
  longueur : **Naturel**
**finstructure**

**fonction** mot () **: Mot**
  **Déclaration** mot : **Mot**
**début**
  mot.contenu ← ”
  mot.longueur ← 0
  **retourner** mot
**fin**

**fonction** genererMot (chaine : **Chaîne de caractères**) **: Mot**
  **Déclaration** mot : **Mot**,
    i : **Entier**
**début**
  mot ← mot()
  **pour** i ←1 **à** longueur(chaine) **faire**
    insererCaractere(mot, iemeCaractere(chaine, i), i)
  **finpour**
  **retourner** mot
**fin**

**fonction** obtenirContenu (mot : **Mot**) **: Chaîne de caractères**
**début**
  **retourner** mot.contenu
**fin**

**fonction** sontEgaux (mot1 : **Mot**, mot2 : **Mot**) **: Booléen**
  **Déclaration** res : **Booléen**
  i : **NaturelNonNul**
**début**
  **si** mot1.longueur ≠ mot2.longueur **alors**
    **retourner** faux
  **sinon**
    res ← vrai
    i ← 1
    **tant que** res ET i ≤ longueur(mot1) **faire**
      res ← iemeCaractere(mot1, i) = iemeCaractere (mot2, i)
      i ← i + 1
    **fintantque**
    **retourner** res
  **finsi**
**fin**
**fonction** longueur (mot : **Mot**) **: Naturel**
**début**
  **retourner** mot.longueur
**fin**

**procédure** insererCaractere (**E/S** mot : **Mot** ; **E** c : **Caractère**, position : **NaturelNonNul**)
  ⌊**précondition(s)** 1 ≤ position ≤ longueur(mot) + 1 ET estLettreOuTiret(c)
  **Déclaration** i : **NaturelNonNul**
**début**
  **pour** i ←longueur(mot) + 1 **à** max(position + 1, 2) **pas de** -1 **faire**
    mot.contenu[i] ← iemeCaractere(mot, i - 1)
  **finpour**
  mot.longueur ← longueur(mot) + 1
  modifierCaractere(mot, c, position)
**fin**

**procédure** modifierCaractere (**E/S** mot : **Mot** ; **E** c : **Caractère**, position : **NaturelNonNul**)
  ⌊**précondition(s)** 1 ≤ position ≤ longueur(mot) ET estLettreOuTiret(c)
**début**
  mot.contenu[position] ← c
**fin**

**procédure** supprimerCaractere (**E/S** mot : **Mot** ; **E** position : **NaturelNonNul**)
  ⌊**précondition(s)** 1 ≤ position ≤ longueur(mot)
  **Déclaration** c : **Caractère**
**début**

```
   si longueur(mot) = 1 ET position = 1 alors
      retourner mot()
   sinon
      pour i ←position à longueur(mot) - 1 faire
         c ← iemeCaractere(mot, i + 1)
         modifierCaractere(mot, c, i)
      finpour
      mot.longueur ← longueur(mot) - 1
   finsi
fin

fonction iemeCaractere (mot : Mot, position : NaturelNonNul) : Caractère
   ⌊précondition(s)  1 ≤ position ≤ longueur(mot)
début
   retourner mot.contenu[position]
fin

fonction concatener (mot1 : Mot, mot2 : Mot) : Mot
   Déclaration  c : Caractère
début
   pour i ←longueur(mot1) + 1 à longueur(mot1) + longueur(mot2) faire
      c ← iemeCaractere(mot2, i - longueur(mot1))
      insererCaractere(mot1, c, i)
      mot1.longueur ← i
   finpour
   retourner mot1
fin

fonction estUnMotValide (mot : Mot) : Booléen
   Déclaration  res : Booléen
                i : NaturelNonNul
début
   i ← 1
   res ← vrai
   tant que res ET i ≤ longueur(mot) faire
      res ← estLettreOuTiret(iemeCaractere(mot, i))
      i ← i + 1
   fintantque
   retourner res
fin

// Sépare en deux mots. L'indice donné est le début du deuxième mot. separerMots(abc, 2) donne a et bc.
procédure separerMots (E mot : Mot, position : NaturelNonNul ; S mot1 : Mot, mot2 : Mot)
   ⌊précondition(s)  1 ≤ position ≤ longueur(mot)
   Déclaration  c : Caractère
                i : NaturelNonNul
début
   mot1 ← mot()
```

```
    pour i ←1 à position - 1 faire
        c ← iemeCaractere(mot, i)
        insererCaractere(mot1, c, i)
        mot1.longueur ← longueur(mot1) + 1
    finpour

    mot2 ← mot()
    pour i ←position à longueur(mot) faire
        c ← iemeCaractere(mot, i)
        insererCaractere(mot2, c, i)
        mot2.longueur ← longueur(mot2) + 1
    finpour
fin

fonction sousMot (mot : Mot, debut : NaturelNonNul, longueur : NaturelNonNul) : Mot
    ⌊précondition(s)  1 ≤ debut + longueur ≤ longueur(mot) et non(longueur(mot) = 0
    Déclaration  c : Caractère
                 i : NaturelNonNul
                 nouveau : Mot
début
    nouveau ← mot()
    pour i ←debut à debut + longueur faire
        c ← iemeCaractere(mot, i)
        insererCaractere(nouveau, c, i - debut + 1)
        nouveau.longueur ← longueur(nouveau) + 1
    finpour
    retourner nouveau
fin

fonction estSousMot (mot1 : Mot, mot2 : Mot) : Booléen
    Déclaration  ssMot : Mot
                 res : Booléen
                 i, j : NaturelNonNul
début
    si longueur (mot1) > longueur(mot2) alors
        retourner faux
    sinon
        pour i ←1 à longueur(mot2) - longueur(mot1) faire
            ssMot ← sousMot(mot2, i, longueur(mot1))
            res ← vrai
            j ← 1
            tant que res ET j ≤ longueur(mot1) faire
                res ← iemeCaractere(mot1, j) = iemeCaractere(ssMot, j)
                j ← j + 1
            fintantque
        finpour
        retourner res
    finsi
```

**fin**

**fonction** sontIdentiques (mot1 : **Mot**, mot2 : **Mot**) **: Booléen**
**début**
  **retourner** (&mot1 = &mot2)
**fin**

## 1.2 TAD SuperMot

**Type** SuperMot **= Structure**
  mot : **Mot**
  *// La position du mot dans la chaîne donnée en entrée du programme.*
  position : **NaturelNonNul**
  listeCorrections : **Liste**<**Mot**>
  *// Indique si le mot est mal orthographié ou non.*
  valide : **Booléen**
**finstructure**

**fonction** superMot **() : SuperMot**
  **Déclaration** supermot : **SuperMot**
**début**
  supermot.mot ← mot()
  supermot.position ← 1
  supermot.listeCorrections ← listeVide()
  supermot.valide ← faux
  **retourner** supermot
**fin**

**fonction** obtenirMot (supermot : **SuperMot**) **: Mot**
**début**
  **retourner** supermot.mot
**fin**

**procédure** fixerMot **(E/S** supermot : **SuperMot**, **E** mot : **Mot)**
**début**
  supermot.mot ← mot
**fin**

**fonction** obtenirPosition (supermot : **SuperMot**) **: NaturelNonNul**
**début**
  **retourner** supermot.position
**fin**

**procédure** fixerPosition **(E/S** supermot : **SuperMot**, **E** position : **NaturelNonNul)**
**début**
  supermot.position ← position
**fin**

**fonction** obtenirListeDeCorrection (supermot : **SuperMot**) **: Liste**<**Mot**>

**début**
  **retourner** supermot.listeCorrections
**fin**

**procédure** fixerListeDeCorrection (**E/S** supermot : **SuperMot**, **E** corrections : **Liste**<**Mot**>)
**début**
  supermot.listeCorrections ← corrections
**fin**

**fonction** obtenirValidite (supermot : **SuperMot**) **: Booléen**
**début**
  **retourner** supermot.valide
**fin**

**procédure** fixerValidite (**E/S** supermot : **SuperMot**, **E** estValide : **Booléen**)
**début**
  supermot.valide ← estValide
**fin**

## 1.3 TAD Dictionnaire

**Type** Dictionnaire = **Structure**
  racines : **Tableau**[1...NBLETTRES] **de ElementDictionnaire**
  longueur : **Naturel**
**finstructure**

**fonction** dictionnaire () **: Dictionnaire**
  **Déclaration** i : **NaturelNonNul**, dictionnaire : **Dictionnaire**
**début**
  **pour** i ←1 **à** NBLETTRES **faire**
    dictionnaire.racines[i] ← **null**
  **finpour**
  dictionnaire.longueur ← 0
  **retourner** dictionnaire
**fin**

**fonction** estVide (dictionnaire : **Dictionnaire**) **: Booléen**
**début**
  **retourner** dictionnaire.longueur = 0
**fin**

**procédure** insererMot (**E/S** dictionnaire : **Dictionnaire**; **E** mot : **Mot**)
  **Déclaration** i, j, indice : **NaturelNonNul**
    longueur : **Naturel**
    tableauDesElementsSuivants : **Tableau**[1...NBLETTRES] **de ElementDictionnaire**
    element, nouvelElement : **ElementDictionnaire**
**début**
  **pour** indice ←1 **à** NBLETTRES **faire**

```
        tableauDesElementsSuivants[indice] ← dictionnaire.racines[indice]
    finpour
    longueur ← dictionnaire.longueur
    pour i ←1 à longueur(mot) faire
        j ← 1
        // On vient chercher dans notre tableau le caractère correspondant au caractère de notre mot
        tant que j ≤ longueur ET obtenirLettre(tableauDesElementsSuivants[j]) ≠ iemeCaractere(mot, i)
        faire
            j ← j + 1
        fintantque
        si j ≤ longueur alors
            element ← tableauDesElementsSuivants[j]
        sinon
            nouvelElement ← element(iemeCaractere(mot, i), faux)
            // La racine n'existe pas dans le dictionnaire, on doit la créer
            si i = 1 alors
                dictionnaire.racines[dictionnaire.longueur] ← nouvelElement
                dictionnaire.longueur ← dictionnaire.longueur + 1
            finsi
            ajouterElementSuivant(element, nouvelElement)
            element ← nouvelElement
        finsi
        // On passe à l'élément suivant pour le prochain caractère du mot
        pour indice ←1 à NBLETTRES faire
            tableauDesElementsSuivants[indice] ← (obtenirTableauDesElementsSuivants(element))[indice]
        finpour
        longueur ← obtenirLongueurTableauDesElementsSuivants(element)
    finpour
    ajouterPresence(element)
fin

procédure insererFichier (E/S dictionnaire : Dictionnaire ; E nomFichier : Chaîne de caractères)
    Déclaration  fichier : FichierTexte,
                 ligne : Chaîne de caractères
début
    fichier ← fichierTexte(nomFichier)
    ouvrir(fichier, lecture)
    pour chaque ligne de fichier
        insererMot(dictionnaire, genererMot(ligne))
    finpour
    fermer(fichier)
fin

fonction estPresent (dictionnaire : Dictionnaire, mot : Mot) : Booléen
    Déclaration  i, j, indice : NaturelNonNul
                 longueur : Naturel
                 tableauDesElementsSuivants : Tableau[1...NBLETTRES] de ElementDictionnaire
                 element : ElementDictionnaire
début
```

```
    pour indice ←1 à NBLETTRES faire
        tableauDesElementsSuivants[indice] ← dictionnaire.racines[indice]
    finpour
    longueur ← dictionnaire.longueur
    // Vérifions que l'on trouve le mot caractère après caractère dans le dictionnaire
    pour i ←1 à longueur(mot) faire
        j ← 1
        tant que j ≤ longueur ET obtenirLettre(tableauDesElementsSuivants[j]) ≠ iemeCaractere(mot, i)
        faire
            j ← j + 1
        fintantque
        // On vérfie si on est sorti du while parce qu'on est arrivé à la fin du tableau ou si on a trouvé la
        bonne lettre
        si j ≤ longueur ET obtenirLettre(tableauDesElementsSuivants[j]) = iemeCaractere(mot, i) alors
            element ← tableauDesElementsSuivants[j]
        sinon
            retourner faux
        finsi
        // Tout va bien, on continue à avancer pour le prochain caractère
        pour indice ←1 à NBLETTRES faire
            tableauDesElementsSuivants[indice] ← (obtenirTableauDesElementsSuivants(element))[indice]
        finpour
        longueur ← obtenirLongueurTableauDesElementsSuivants(element)
    finpour
    // Si on n'a eu toujours aucune erreur, on retourne la valeur du booléen du dernier caractère
    retourner obtenirBooleen(element)
fin

fonction charger (nomFichier : Chaîne de caractères) : Dictionnaire
    Déclaration  fichier : FichierTexte
                 dictionnaire : Dictionnaire
                 longueur : Naturel
                 i : NaturelNonNul
début
    dictionnaire ← dictionnaire()
    fichier ← fichierTexte(nomFichier)
    ouvrir(fichier, lecture)
    longueur ← recupererUneLongueur(fichier)
    dictionnaire.longueur ← longueur
    pour i ←1 à longueur faire
        dictionnaire.racines[i] ← chargerElementsSuivants(fichier)
    finpour
    fermer(fichier)
    retourner dictionnaire
fin

fonction chargerElementsSuivants (fichier : FichierTexte) : ElementDictionnaire
    Déclaration  caractere : Caractère
                 booleen : Booléen
```

```
                longueur : Naturel
                element : ElementDictionnaire
                i : NaturelNonNul
début
   caractere ← recupererUnCaractere(fichier)
   booleen ← recupererUnBooleen(fichier)
   longueur ← recupererUneLongueur(fichier)
   element ← element(caractere, booleen)
   pour i ←1 à longueur faire
      ajouterElementSuivant(element, chargerElementsSuivants(fichier))
   finpour
   retourner element
fin

fonction sauvegarder (dictionnaire : Dictionnaire, emplacement : Chaîne de caractères) : Fichier-
Texte
   Déclaration  fichier : FichierTexte
                dizaine, unite : Caractère
                i : NaturelNonNul
début
   fichier ← fichierTexte(emplacement)
   ouvrir(fichier, ecriture)
   naturelEnCaracteres(dictionnaire.longueur, dizaine, unite)
   si dizaine ≠ '0' alors
      ecrireCaractere(fichier, dizaine)
   finsi
   ecrireCaractere(fichier, unite)
   pour i ←1 à dictionnaire.longueur faire
      ecrireCaractere(fichier, obtenirLettre(dictionnaire.racines[i]))
      ecrireCaractere(fichier, booleenEnCaractere(obtenirBooleen(dictionnaire.racines[i])))
      naturelEnCaracteres(obtenirLongueurTableauDesElementsSuivants(dictionnaire.racines[i]), dizaine, unite)

      si dizaine ≠ '0' alors
         ecrireCaractere(fichier, dizaine)
      finsi
      ecrireCaractere(fichier, unite)
      sauvegarderElementsSuivants(fichier, dictionnaire.racines[i])
   finpour
   fermer(fichier)
   retourner fichier
fin

procédure sauvegarderElementsSuivants (E/S fichier : FichierTexte ; E elementDico : ElementDic-
tionnaire)
   Déclaration  tableauElementsSuivants : Tableau[1...NBLETTRES] de ElementDictionnaire
                dizaine, unite : Caractère
                i, indice : NaturelNonNul
début
   pour indice ←1 à NBLETTRES faire
```

```
    tableauDesElementsSuivants[indice] ← (obtenirTableauDesElementsSuivants(elementDico))[indice]
  finpour
  pour i ←1 à obtenirLongueurTableauDesElementsSuivants(elementDico) faire
    ecrireCaractere(fichier, obtenirLettre(tableauElementsSuivants[i]))
    ecrireCaractere(fichier, booleenEnCaractere(obtenirBooleen(tableauElementsSuivants[i])))
    naturelEnCaracteres(obtenirLongueurTableauDesElementsSuivants(tableauElementsSuivants[i]), di-
    zaine, unite)
    si dizaine ≠ '0' alors
      ecrireCaractere(fichier, dizaine)
    finsi
    ecrireCaractere(fichier, unite)
    sauvegarderElementsSuivants(fichier, tableauElementsSuivants[i])
  finpour
fin
```

## 1.4 TAD ElementDictionnaire

```
Type ElementDictionnaire = ^NoeudDictionnaire

Type NoeudDictionnaire = Structure
  lettre : Caractère
  presence : Booléen
  longueur : NaturelNonNul
  tableauDesElementsSuivants : Tableau[1...NBLETTRES] de ^ElementDictionnaire
finstructure

fonction element (c : Caractère, b : Booléen) : ElementDictionnaire
  Déclaration  elementDico : ElementDictionnaire
début
  allouer(elementDico)
  elementDico^.lettre ← c
  elementDico^.presence ← b
  elementDico^.longueur ← 0
  pour i ←1 à NBLETTRES faire
    elementDico^.tableauDesElementsSuivants ← null
  finpour
  retourner elementDico
fin

procédure ajouterElementSuivant (E/S elementDico : ElementDictionnaire ; E nouvelElementDico :
ElementDictionnaire)
  Déclaration  indice :NaturelNonNul
début
  elementDico^.longueur ← elementDico^.longueur + 1
  indice ← elementDico^.longueur
  elementDico^.tableauDesElementsSuivants[indice] ← nouvelElementDico
fin
```

**fonction** obtenirTableauDesElementsSuivants (**E** elementDico : **ElementDictionnaire**) : **Tableau**[1...NBLET
**de** ^**ElementDictionnaire**
**début**
  **retourner** elementDico^.tableauDesElementsSuivants
**fin**

**fonction** obtenirLongueurTableauDesElementsSuivants (**E** elementDico : **ElementDictionnaire**) : **Naturel**
**début**
  **retourner** elementDico^.longueur
**fin**

**fonction** obtenirLettre (elementDico : **ElementDictionnaire**) : **Caractère**
**début**
  **retourner** elementDico^.lettre
**fin**

**fonction** obtenirBooleen (elementDico : **ElementDictionnaire**) : **Booléen**
**début**
  **retourner** elementDico^.presence
**fin**

**procédure** supprimerElementSuivant (**E/S** elementDico : **ElementDictionnaire** ; **E** indice : **NaturelNonNul**)
  ⌊**précondition(s)** 1 ≤ indice ≤ obtenirLongueurTableauDesElementsSuivants(elementDico)
  **Déclaration** i, j :**NaturelNonNul**
**début**
  *// On supprime tous les fils de l'élément que l'on veut supprimer*
  **pour** j ←1 **à** obtenirLongueurTableauDesElementsSuivants(elementDico^.tableauDesElementsSuivants[indice
  **faire**
    supprimerElementSuivant(elementDico^.tableauDesElementsSuivants[indice], j)
  **finpour**

  *// On supprime l'élément que l'on veut supprimer et on décrémente la longueur du père*
  **desallouer**(elementDico^.tableauDesElementsSuivants[indice])
  elementDico^.longueur ← elementDico^.longueur - 1

  *// On supprime le pointeur (qui ne pointe plus vers rien) du tableau du père en décalant*
  **pour** i ←indice **à** elementDico^.longueur **faire**
    elementDico^.tableauDesElementsSuivants[i] ← elementDico^.tableauDesElementsSuivants[i+1]
  **finpour**
**fin**

## 1.5 TAD Correcteur Orthographique

**fonction** proposerMots (dico : **Dictionnaire**, mot : **Mot**) : **Liste**<**Mot**>
  **Déclaration**
**début**

```
    retourner motsExacts(possibilitesMots(mot), dico)
fin

fonction possibilitesMot (mot : Mot) : Liste<Mot>
    Déclaration  liste, liste2, liste3, liste4 : Liste<Mot>
                 i : Naturel
début
    i ← 0
    liste ← genererMotParRemplacement(mot)
    liste2 ← genererMotParSuppression(mot)
    liste3 ← genererMotParInsertion(mot)
    liste4 ← enererMotsParInversion(mot)
    pour i ←1 à longueur(liste2) faire
        ajouter(liste, obtenirElement(liste2, i))
    finpour
    pour i ←1 à longueur(liste3) faire
        ajouter(liste, obtenirElement(liste3, i))
    finpour
    pour i ←1 à longueur(liste4) faire
        ajouter(liste, obtenirElement(liste4, i))
    finpour
    retourner liste
fin

fonction enererMotsParInversion (mot : Mot) : Liste<Mot>
    Déclaration  liste : Liste<Mot>
                 motTemp : Mot
                 c : Caractère
                 i : NaturelNonNul
début
    liste ← liste()
    pour i ←1 à longueur(mot) faire
        motTemp ← mot
        c ← iemeCaractere(mot, i)
        modifierCaractere(motTemp, iemeCaractere(mot, i + 1, i))
        modifierCaractere(motTemp, c, i + 1)
        inserer(liste, i, motTemp
    finpour
    retourner liste
fin

fonction genererMotsParSuppression (mot : Mot) : Liste<Mot>
    Déclaration  liste : Liste<Mot>
                 motTemp : Mot
                 i : NaturelNonNul
début
    liste ← liste()
    pour i ←1 à longueur(mot) faire
        motTemp ← mot
```

```
        supprimerCaractere(motTemp, i)
        inserer(liste, i, motTemp)
    finpour
    retourner liste
fin

fonction genererMotsParRemplacement (mot : Mot) : Liste<Mot>
    Déclaration  lettre : Tableau[1..NBLETTRES]  de Caractère
                 liste : Liste<Mot>
                 motTemp : Mot
début
    liste ← liste()
    lettre[1] ← 'a'
    pour i ←1 à NBLETTRES - 1 faire
        lettre[i+1] ← succ(lettre[i])
    finpour
    pour i ←1 à longueur(mot) faire
        motTemp ← mot
        pour j ←1 à NBLETTRES faire
            modifierCaractere(motTemp, lettre[j], i)
            inserer(liste, 1, motTemp)
        finpour
    finpour
    retourner liste
fin

fonction genererMotsParInsertion (mot : Mot) : Liste<Mot>
    Déclaration  lettre : Tableau[1..NBLETTRES]  de Caractère
                 liste : Liste<Mot>
                 motTemp : Mot
début
    liste ← liste()
    lettre[1] ← 'a'
    pour i ←1 à NBLETTRES - 1 faire
        lettre[i+1] ← succ(lettre[i])
    finpour
    pour i ←1 à longueur(mot) + 1 faire
        pour j ←1 à NBLETTRES faire
            motTemp ← mot
            insererCaractere(motTemp, lettre[j], i)
            inserer(liste, 1, motTemp)
        finpour
    finpour
    retourner liste
fin

fonction motsExacts (liste : Liste<Mot>, d : Dictionnaire) : Liste<Mot>
    Déclaration  mot : Mot
                 listeFinale : Liste
```

```
                i : NaturelNonNul
début
   listeFinale ← liste()
   pour i ←1 à longueur(liste) faire
      mot ← obtenirElement(liste, i)
      si estDansLeDictionnaire(dictionnaire, mot)) alors
         ajouter(listeFinale, mot)
      finsi
   finpour
   retourner listeFinale
fin
```

## 1.6 TAD Parametres

```
// Structure permettant de manipuler les paramètres reçus par le programme (-h, -d, -f, -malandain etc etc ..)
Type Parametre = Structure
   // variable initialisée si il y a un '-d' dans les paramètres, elle contient le nom du dictionnaire
   dictionnaire : Chaîne de caractères
   // variable initialisée si il y a un '-f' dans les paramètres, elle contient le nom du fichier
   fichier : Chaîne de caractères
   // La chaîne de caractères donnée en entrée à corriger
   cible : Chaîne de caractères
   // Variables qui prend vrai SSI il y a un '-h' dans les paramètres.
   aide : Booléen
finstructure
fonction parametres () : Parametres
   Déclaration   p : Parametres
début
   p.dictionnaire ← NULL
   p.fichier ← NULL
   p.cible ← NULL
   p.aide ← Faux

fin

procédure fixerDictionnaire (E/S p : Parametres, E c : Chaîne de caractères)
début
   p.dictionnaire ← c
fin

procédure fixerFichier (E/S p : Parametres, E c : Chaîne de caractères)
début
   p.fichier ← c
fin

procédure fixerCible (E/S p : Parametres, E c : Chaîne de caractères)
début
   p.cible ← c
```

**fin**

**procédure** fixerAide (**E/S** p : **Parametres**, **E** b : **Booléen**)
**début**
  p.aide ← b
**fin**

**fonction** obtenirDictionnaire (p : **Parametres**) **: Chaîne de caractères**
**début**
  **retourner** p.dictionnaire
**fin**

**fonction** obtenirFichier (p : **Parametres**) **: Chaîne de caractères**
**début**
  **retourner** p.fichier
**fin**

**fonction** obtenirAide (p : **Parametres**) **: Booléen**
**début**
  **retourner** p.aide
**fin**

**fonction** obtenirCible (p : **Parametres**) **: Chaîne de caractères**
**début**
  **retourner** p.cible
**fin**

# Chapitre 2

# Conception détaillée de la logique métier

**procédure** asispell (**E** argc : **Naturel**, argv : **Tableau**[1..nbParametres] **de Chaîne de caractères**)
  **Déclaration** p : **Parametres**
**début**
  p ← formaterParametres(argc, argv)
  executerProgramme(p)
**fin**

**fonction** formaterParametres (nbParametre : **Entier**, parametres : **Tableau**[1..nbParametres] **de Chaîne de caractères**) **: Parametres**
  **Déclaration** optch : **Entier**
    format : **Chaîne de caractères**
    p : **Parametres**
**début**
  *// un paramère simple : la lettre*
  *// un paramètre avec argument : la lettre suivi de " :"*
  format ← "hd :f :"
  p ← parametres()
  *// fonction équivalente au getopt() en C*
  **tant que**
  optch ← parseLesParametres(argc, argv, format) ≠ -1 **faire**
    **si** optch = 'h' **alors**
      fixerAide(p, vrai)
    **finsi**
    **si** optch = 'f' **alors**
      fixerFichier(p, optarg)
    **finsi**
    **si** optch = 'd' **alors**
      fixerDictionnaire(p, optarg)
    **finsi**
  **fintantque**
  **retourner** parametres
**fin**

**procédure** executerProgramme (**E** parametres : **Parametres**)
  **Déclaration** d : **Dictionnaire**
    entreeStandard : **Chaîne de caractères**

```
début
   d ← dictionnaire()
   si obtenirAide(parametres) alors
      afficherAide()
   sinon
      // Si on a un fichier et un dictionnaire, on complète le dictionnaire déjà sérialisé
      si (obtenirFichier(parametres) ≠ NULL) ET (obtenirDictionnaire(parametres) ≠ NULL) alors
         d ← charger(obtenirDictionnaire(parametres))
         insererFichier(d, obtenirFichier(parametres))
         sauvegarder(d, obtenirDictionnaire(parametres))
      sinon
         entreeStandard ← lire()
         fixerCible(parametres, entreeStandard)
         // On effectue la correction avec le dictionnaire déjà sérialisé
         si (obtenirCible(parametres) ≠ NULL) ET (obtenirDictionnaire(parametres)) ≠ NULL alors
            d ← charger(obtenirDictionnaire(parametres))
            corriger(obtenirCible(parametres), d)
         finsi
      finsi
   finsi
fin

procédure ajouterDansLeDictionnaire (E/S dictionnaire : Dictionnaire ; E cheminDuFichier :Chaîne de caractères)
début
   insererFichier(dictionnaire, cheminDuFichier)
fin

fonction corriger (chaine : Chaîne de caractères, dictionnaire : Dictionnaire) : Chaîne de caractères
   Déclaration  chaineValide : Booléen
                erreur : Entier
                listeDeSuperMot : Liste<SuperMot>
                resultat : Chaîne de caractères
début
   evaluerSaisie(chaineValide, erreur, chaine)
   resultat ← ""
   si chaineValide alors
      listeSuperMot ← genererListeDeSuperMots(chaine, dictionnaire)
      corrigerListeDeSuperMots(listeDeSuperMots, dictionnaire)
      afficherCorrection(listeDeSuperMots)
   finsi
   ( afficherResultat(erreur, chaine))
fin

procédure evaluerSaisie (E/S estValide : Booléen, erreur : Entier ;E chaine : Chaîne de caractères)
   Déclaration  i : NaturelNonNul

début
```

```
    i ← 1
    estValide^ ← 1
    tant que i ≤ longueur(chaine) ET estValide^ faire
        si non(estCaractereValide(iemeCaractere(chaine, i)) alors
            estValide^ ← 0
            erreur^ ← 1
        finsi
        i ← i + 1
    fintantque
fin

fonction genererListeDeSuperMots (chaine : Chaîne de caractères, dictionnaire : Dictionnaire) :
Liste<SuperMot>
    Déclaration
début
    retourner separerMots(chaine, dictionnaire)
fin

fonction separerMots (chaine : Chaîne de caractères, dictionnaire : Dictionnaire) : Liste<SuperMot>

    Déclaration  i : NaturelNonNul
                 listeRetour : Liste<SuperMot>
                 motTemp : Mot
                 superMotTemp : SuperMot
début
    motTemp ← mot()
    listeRetour ← liste()
    pour i ←1 à longueur(chaine) + 1 faire
        si estLettreOuTiret(iemeCaractere(chaine, i)) alors
            insererCaractere(motTemp, iemeCaractere(chaine, i), longueur(motTemp)+1)
        sinon
            si longueur(motTemp) ≠ 0 alors
                fixerMot(superMotTemp, motTemp)
                fixerPosition(superMotTemp,i - (longueur(motTemp) + 1))
                fixerValidite(superMotTemp, verifieValidite(motTemp, dictionnaire))
                inserer(listeRetour, 1, superMotTemp)
            finsi
            motTemp ← mot()
        finsi
    finpour
    retourner listeRetour
fin

fonction verifieValidite (mot : Mot, dictionnaire Dictionnaire) : Booléen
    Déclaration
début
    retourner estPresent(dictionnaire, mot)
fin
```

```
procédure corrigerListeSuperMot (E/S listeDeSuperMots : Liste<SuperMot> ; E dictionnaire : Dictionnaire)
   Déclaration  i : Entiersuper MotTemp : SuperMot
début
   pour i ←1 à longueur(listeDeSuperMots) faire
      superMotTemp ← obtenirElement(listeDeSuperMots, i)
      si non superMotTemp.estValide alors
         superMotTemp.listeDeCorrections ← proposerMot(superMotTemp.mot, dictionnaire)
      finsi
   finpour
fin

fonction afficherCorrection (listeDeSuperMots : Liste<SuperMot>) : Chaîne de caractères
   Déclaration  i, j : Entier
                superMotTemp : SuperMot
                chaine : Chaîne de caractères
début
   chaine ← ""
   pour i ←1 à longueur(listeDeSuperMots) faire
      superMotTemp ← obtenirElement(listeDeSuperMots, i)
      si superMotTemp.estValide = VRAI alors
         chaine ← chaine + "*"
      sinon
         chaine ← chaine + "&_ " + obtenirContenu(superMotTemp.mot) + "_"
         chaine ← chaine + entierEnChaine(longueur(superMotTemp.listeCorrections)) + "_"
         chaine ← chaine + entierEnChaine(superMotTemp.position) + " :_"
         chaine ← chaine + listeDeMotsEnChaine(superMotTemp.listeCorrections)
      finsi
      chaine ← chaine + retout à la ligne
   finpour
   ecrire(chaine)
fin

fonction listeDeMotsEnChaine (listeDeMots : Liste<Mot>) : Chaîne de caractères
   Déclaration  i : Entier
                chaine : Chaîne de caractères
                motTemp : Mot
début
   chaine ← ""
   pour j ←1 à longueur(listeDeMots) faire
      motTemp ← obtenirElement(listeDeMots, j)
      chaine ← chaine + "_" + obtenirContenu(motTemp)
   finpour
   retourner chaine
fin

procédure afficherResultat (E erreur : Entier, chaine : Chaîne de caractères)
début
   si erreur = 0 alors
```

```
        afficherCorrection(chaine)
    sinon
        afficherErreur(erreur)
    finsi
fin

procédure afficherErreur (E typerDErreur : Entier)
début
    si typerDErreur = 1 alors
        ecrire(Au moins un caractère entré n'est pas valide.)
    sinon
        si typerDErreur = 2 alors
            ecrire(Type d'erreur 2)
        finsi
    finsi
fin

procédure afficherCorrection (E correction : Chaîne de caractères)
début
    ecrire(correction)
fin

procédure afficherAide ()
début
    ecrire('Aide asispell :')
    ecrire(' asispell -h : cette aide')
    ecrire(' asispell -d dico : correction de l'entrée standard à l'aide du dictionnaire dico)
    ecrire(' asispell -d dico - fic : compléter le dictionnaire dico à l'aide du fichier fic)
fin
```

# Quatrième partie

# Code C et tests unitaires

# Chapitre 1

# Fichiers headers des TAD

## Sommaire



## 1.1 CorrecteurOrthographique

### 1.1.1 Public

```
1  /**
2   * \file TADCorrecteurOrthographique.h
3   * \brief Partie privée : implantation du TAD CorrecteurOrthographique avec la SDD Correct
4   * \version 1.0
5   */
6  #ifndef __TAD_CORRECTEUR_ORTHOGRAPHIQUE__
7  #define __TAD_CORRECTEUR_ORTHOGRAPHIQUE__
8  #include "TADMot.h"
9  #include "TADDictionnaire.h"
10 #include "TADListeDeMot.h"
```

```
11
12 /**
13  * \fn ListeDeMot CorrecteurOrthographique_proposerMots(Dictionnaire dictionnaire, Mot mot,
14  * \brief Récupère des corrections pour un mot
15  * \param Mot mot : Le mot à corriger
16  * \return Retourne une liste de mots correspondant aux corrections trouvées pour le param
17  * \author Étienne Batise
18  */
19 ListeDeMot CorrecteurOrthographique_proposerMots(Dictionnaire dictionnaire, Mot mot);
20 #endif
```

### 1.1.2 Privé

```
1  #ifndef __CORRECTEUR_ORTHOGRAPHIQUE_PRIVE__
2  #define __CORRECTEUR_ORTHOGRAPHIQUE_PRIVE__
3  #include "TADMot.h"
4  #include "TADDictionnaire.h"
5  #include "TADListeDeMot.h"
6
7  /**
8  * \fn ListeDeMot CorrecteurOrthographique_possibiltesMot(Mot mot);
9  * \brief Récupère des mots pouvant être construit à partir d'un mot
10 * \param Mot mot : le mot en question
11 * \return ListeDeMot
12 * \author Étienne Batise
13 */
14 ListeDeMot CorrecteurOrthographique_possibilitesMots(Mot mot);
15 /**
16 * \fn ListeDeMot CorrecteurOrthographique_genererMotsParSuprresion(Mot mot);
17 * \brief Récupère des mots pouvant être construit à partir d'un mot en enlevant une lettre
18 * \param Mot mot : le mot en question
19 * \return ListeDeMot
20 * \author Étienne Batise
21 */
22 ListeDeMot CorrecteurOrthographique_genererMotsParSuppression(Mot mot);
23
24 /**
25 * \fn ListeDeMot CorrecteurOrthographique_genererMotsParSuprresion(Mot mot);
26 * \brief Récupère des mots pouvant être construit à partir d'un mot en remplaçant une lett
27 * \param Mot mot : le mot en question
28 * \return ListeDeMot
29 * \author Étienne Batise
30 */
31 ListeDeMot CorrecteurOrthographique_genererMotsParRemplacement(Mot mot);
32
33 /**
34 * \fn ListeDeMot CorrecteurOrthographique_genererMotsParSuprresion(Mot mot);
35 * \brief Récupère des mots pouvant être construit à partir d'un mot en insérant une lettre
36 * \param Mot mot : le mot en question
37 * \return ListeDeMot
```

```
38  * \author Étienne Batise
39  */
40  ListeDeMot CorrecteurOrthographique_genererMotsParInsertion(Mot mot);
41
42  /**
43  * \fn ListeDeMot CorrecteurOrthographique_genererMotsParInversion(Mot mot);
44  * \brief Récupère des mots pouvant être construit à partir d'un mot en inversant des lettr
45  * \param Mot mot : le mot en question
46  * \return ListeDeMot
47  * \author Antoine Augusti
48  */
49  ListeDeMot CorrecteurOrthographique_genererMotsParInversion(Mot mot);
50
51  /**
52  * \fn ListeDeMot CorrecteurOrthographique_motsExacts(ListeDeMot listeDeMots, Dictionnaire
53  * \param ListeDeMot listeDeMots : la liste de Mot à filtrer
54  * \param Dictionnaire dictionnaire : le dictionnaire qui sert de référence
55  * \author Étienne Batise
56  */
57  ListeDeMot CorrecteurOrthographique_motsExacts(ListeDeMot listeDeMots, Dictionnaire dictionna
58
59  #endif
```

## 1.2 Dictionnaire

```
1   /**
2    * \file TADDictionnaire.h
3    * \brief Implantation du TAD Dictionnaire avec la SDD Dictionnaire
4    * \version 1.0
5    */
6   #ifndef __TAD_DICTIONNAIRE__
7   #define __TAD_DICTIONNAIRE__
8   #include "TADElementDictionnairePrive.h"
9   #include "TADMot.h"
10  #include "TADFichierTexte.h"
11
12  typedef struct {
13      unsigned int longueur;
14      ElementDictionnaire racines[NB_LETTRES];
15  } Dictionnaire;
16
17  /**
18   * \fn Dictionnaire Dictionnaire_dictionnaire();
19   * \author Jean-Claude Bernard
20   * \brief Crée un Dictionnaire
21   * \return retourne un dictionnaire vide
22   */
23  Dictionnaire Dictionnaire_dictionnaire();
24
```

```c
/**
 * \fn unsigned int Dictionnaire_estVide(Dictionnaire dictionnaire);
 * \author Antoine Augusti
 * \brief Vérifie si un dictionnaire est vide
 * \param Dictionnaire dictionnaire : le dictionnaire que l'on veut tester
 * \return retourne un booléen (unsigned int), 1 si il est vide, 0 sinon
 */
unsigned int Dictionnaire_estVide(Dictionnaire dictionnaire);

/**
 * \fn void Dictionnaire_insererMot(Dictionnaire* dictionnaire, Mot mot);
 * \author Jean-Claude Bernard
 * \brief Insère un mot dans un dictionnaire
 * \param Dictionnaire* dictionnaire : le dictionnaire auquel on veut insérer le mot
 * \param Mot mot : le mot que l'on veut ajouter
 */
void Dictionnaire_insererMot(Dictionnaire* dictionnaire, Mot mot);

/**
 * \fn void Dictionnaire_insererFichier(Dictionnaire* dictionnaire, char* nomFichier);
 * \author Jean-Claude Bernard
 * \brief Rempli le dictionnaire à partir d'un fichier contenant des mots
 * \param Dictionnaire* dictionnaire : le dictionnaire que l'on veut remplir
 * \param char* nomFichier : le nom du fichier à insérer
 */
void Dictionnaire_insererFichier(Dictionnaire* dictionnaire, char* nomFichier);

/**
 * \fn unsigned int Dictionnaire_estPresent(Dictionnaire dictionnaire, Mot mot);
 * \author Antoine Augusti
 * \brief Vérifier la présence d'un mot dans un dictionnaire
 * \param Dictionnaire dictionnaire : le dictionnaire que l'on veut vérifier
 * \param Mot mot : le mot à trouver
 * \return retourne un booléen
 */
unsigned int Dictionnaire_estPresent(Dictionnaire dictionnaire, Mot mot);

/**
 * \fn Dictionnaire Dictionnaire_charger(char* emplacement);
 * \author Antoine Augusti
 * \brief Charger un dictionnaire depuis un fichier texte où le dictionnaire est sérialisé
 * \param char* emplacement : le nom du fichier contenant le dictionnaire
 * \return retourne un dictionnaire
 */
Dictionnaire Dictionnaire_charger(char* emplacement);

/**
 * \fn Fichier Dictionnaire_sauvegarder(Dictionnaire dictionnaire, char* emplacement);
 * \author Jean-Claude Bernard
 * \brief Sauvegarder un dictionnaire
```

```
 * \param Dictionnaire dictionnaire : le dictionnaire à sauvegarder
 * \param char* emplacement : le nom du fichier dans lequel on veut sauver le dictionnaire
 * \return retourne un fichier
 */
Fichier Dictionnaire_sauvegarder(Dictionnaire dictionnaire, char* emplacement);

#endif
```

## 1.3 ElementDictionnaire

### 1.3.1 Privé

```
#ifndef __TAD_ELEMENT_DICTIONNAIRE__
#define __TAD_ELEMENT_DICTIONNAIRE__

#define NB_LETTRES 96

typedef struct NoeudDictionnaire* ElementDictionnaire;

typedef struct NoeudDictionnaire {
  char lettre;
  unsigned int presence;
  unsigned int longueur;
  ElementDictionnaire tableauDesElementsSuivants[NB_LETTRES];
} NoeudDictionnaire;


/**
 * \fn ElementDictionnaire ElementDico_element(char caractere, unsigned int estPresent);
 * \author Antoine Augusti
 * \brief Crée un ElementDictionnaire
 * \param char caractere : la lettre concernée
 * \param unsigned int estPresent : indique si la lettre est présente ou non
 * \return retourne l'ElementDictionnaire créé
 */
ElementDictionnaire ElementDico_element(char caractere, unsigned int estPresent);

/**
 * \fn void ElementDico_ajouterElementSuivant(ElementDictionnaire *e1, ElementDictionnaire
 * \author Jean-Claude Bernard
 * \brief Ajoute un ElementDictionnaire suivant à l'ElementDictionnaire e1
 * \param ElementDictionnaire e1 : le premier ElementDictionnaire auquel on va ajouter e2
 * \param ElementDictionnaire e2 : l'ElementDictionnaire à ajouter
 */
void ElementDico_ajouterElementSuivant(ElementDictionnaire *e1, ElementDictionnaire e2);

/**
 * \fn void ElementDico_supprimerElementSuivant(ElementDictionnaire *e1, unsigned int indi
 * \author Antoine Augusti
```

```
38  * \brief Supprime l'ElementDictionnaire suivant donné, selon l'indice donné
39  * \param ElementDictionnaire e1 : l'ElementDictionnaire où l'on va supprimer le pointeur
40  * \param unsigned int indice : l'indice du pointeur vers ElementDictionnaire à supprimer
41  * \warning assert(0 <= indice && indice < ElementDico_obtenirLongueurTableauDesElementsSu
42  */
43 void ElementDico_supprimerElementSuivant(ElementDictionnaire *e1, unsigned int indice);
44
45 /**
46  * \fn void ElementDico_ajouterPresence(ElementDictionnaire *e1, unsigned int presence);
47  * \author Jean-Claude Bernard
48  * \brief Fixe la présence de l'ElementDictictionnaire donné
49  * \param ElementDictionnaire e1 : l'ElementDictionnaire où l'on va fixer la présence
50  * \param unsigned int presence : le booleen vrai(1) ou faux(0)
51  */
52 void ElementDico_ajouterPresence(ElementDictionnaire *e1, unsigned int presence);
53
54 /**
55  * \fn ElementDictionnaire (*ElementDico_obtenirTableauDesElementsSuivants(ElementDictionn
56  * \author Jean-Claude Bernard
57  * \brief Retourne le tableau des ElementDictionnaire suivants
58  * \param ElementDictionnaire element : l'ElementDictionnaire auquel on s'intéresse
59  * \param ElementDictionnaire tableauDesElementsSuivants[NB_LETTRES] : le tableau des Elem
60  */
61 ElementDictionnaire (*ElementDico_obtenirTableauDesElementsSuivants(ElementDictionnaire eleme
62
63 /**
64  * \fn unsigned int ElementDico_obtenirLongueurTableauDesElementsSuivants(ElementDictionna
65  * \author Antoine Augusti
66  * \brief Retourne la longueur du tableau des ElementDictionnaire suivants
67  * \param ElementDictionnaire element : l'ElementDictionnaire auquel on s'intéresse
68  * \return la taille du tableau des ElementDictionnaire suivants
69  */
70 unsigned int ElementDico_obtenirLongueurTableauDesElementsSuivants(ElementDictionnaire eleme
71
72 /**
73  * \fn char ElementDico_obtenirLettre(ElementDictionnaire element);
74  * \author Antoine Augusti
75  * \brief Retourne la lettre de l'ElementDictionnaire actuel
76  * \param ElementDictionnaire element : l'ElementDictionnaire auquel on s'intéresse
77  * \return la lettre de l'ElementDictionnaire courant
78  */
79 char ElementDico_obtenirLettre(ElementDictionnaire element);
80
81 /**
82  * \fn unsigned int ElementDico_obtenirBooleen(ElementDictionnaire element);
83  * \author Antoine Augusti
84  * \brief Indique si la lettre est présente pour l'ElementDictionnaire actuel
85  * \param ElementDictionnaire element : l'ElementDictionnaire auquel on s'intéresse
86  * \return vrai si présent, faux sinon
87  */
```

```
88 unsigned int ElementDico_obtenirBooleen(ElementDictionnaire element);
89
90 #endif
```

## 1.4 FichierTexte

```
1  /**
2   * \file TADFichierTexte.h
3   * \brief Implantation du TAD FichierTexte avec la SDD FichierTexte
4   * \version 1.0
5   */
6  #ifndef __TAD_FICHIER_TEXTE__
7  #define __TAD_FICHIER_TEXTE__
8
9  typedef enum {LECTURE, ECRITURE} Mode;
10
11 typedef struct Fichier {
12     FILE* file;
13     char* nom;
14     Mode mode;
15 } Fichier;
16
17 /**
18 * \fn Fichier FichierTexte_fichierTexte(char* nomDuFichier);
19 * \brief création d'un fichier texte à partir d'un fichier identifé par son nom
20 * \param char* nomDuFichier : le nom du fichier
21 * \return Ficher : le fichier
22 * \author Étienne Batise
23 */
24 Fichier FichierTexte_fichierTexte(char* nomDuFichier);
25
26 /**
27 * \fn unsigned int FichierTexte_estOuvert(Fichier fichier);
28 * \brief Permet de savoir si un fichier est ouvert ou non
29 * \param Fichier fichier : le fichier a ouvrir
30 * \return unsigned int : booléen qui vaut vrai si le fichier est ouvert, faux sinon
31 * \author Étienne Batise
32 */
33 unsigned int FichierTexte_estOuvert(Fichier fichier);
34
35 /**
36 * \fn void FichierTexte_ouvrir(Fichier *fichier, Mode mode);
37 * \brief Ouvre un fichier texte en lecture ou en écriture. Si le mode est écriture et que
38 * \param Fichier* fichier : le fichier à ouvrir
39 * \param Mode mode : le mode d'ouverture souhaité
40 * \author Étienne Batise
41 */
42 void FichierTexte_ouvrir(Fichier *fichier, Mode mode);
43
```

```c
/**
* \fn void FichierTexte_fermer(Fichier *fichier);
* \brief Ferme un fichier texte
* \param Fichier fichier : le fichier à fermer
* \author Étienne Batise
*/
void FichierTexte_fermer(Fichier *fichier);


/**
* \fn Mode FichierTexte_mode(Fichier fichier);
* \brief Permet de connaitre le mode d'ouverture d'un fichier
* \param Fichier fichier : le fichier que l'on étudie
* \return Le mode d'ouverture du fichier
* \author Étienne Batise
*/
Mode FichierTexte_mode(Fichier fichier);

/**
* \fn unsigned int FichierTexte_finFichier(Fichier fichier);
* \brief Permet de savoir si l'on est à la fin d'un fichier
* \param Fichier fichier : Le ficher que l'on veut étudier
* \return Un booléen qui vaut 1 si on est à la fin, 0 sinon
* \warning (fichier.mode == LECTURE) && FichierTexte_estOuvert(fichier)
* \author Étienne Batise
*/
unsigned int FichierTexte_finFichier(Fichier fichier);

/**
* \fn void FichierTexte_ecrireChaine(Fichier *fichier, char* chaine);
* \brief Écrit une chaîne suivi d'un retour à la ligne à partir de la position courante du
* \param Fichier* fichier, char* chaine
* \warning FichierTexte_estOuvert(*fichier) && (FichierTexte_mode(*fichier) == ECRITURE)
* \author Étienne Batise
*/
void FichierTexte_ecrireChaine(Fichier *fichier, char* chaine);

/**
* \fn char* FichierTexte_lireChaine(Fichier fichier);
* \brief Lit une chaîne (jusqu'à un retour à la ligne ou à la fin du fichier) à partir de
* \param Fichier fichier
* \return char*  : la chaine lue
* \warning FichierTexte_estOuvert(fichier) && (FichierTexte_mode(fichier) == LECTURE) && !
* \note : Penser à liberer l'espace mémoire alloué à la chaine de caractère
* \author Étienne Batise
*/
char* FichierTexte_lireChaine(Fichier fichier);

/**
* \fn void FichierTexte_ecrireCaractere(Fichier *fichier, char caractere);
```

```
94  * \brief Écrit un caractère à partir de la position courante du fichier
95  * \param Fichier* fichier : le fichier dans lequel on veut écrire un caractère
96  * \param char caractere : le caractère que l'on veut écrire
97  * \warning FichierTexte_estOuvert(*fichier) && (FichierTexte_mode(*fichier) == ECRITURE)
98  * \author Étienne Batise
99  */
100 void FichierTexte_ecrireCaractere(Fichier *fichier, char caractere);
101
102 /**
103 * \fn char FichierTexte_lireCaractere(Fichier fichier);
104 * \brief Lit un caractère à partir de la position courante du fichier
105 * \param Fichier fichier : le fichier dans lequel on veut lire le caractère
106 * \return char : le caractère lut
107 * \warning FichierTexte_estOuvert(fichier) && (FichierTexte_mode(fichier) == LECTURE) && !
108 * \author Étienne Batise
109 */
110 char FichierTexte_lireCaractere(Fichier fichier);
111
112 /**
113 * \fn void FichierTexte_ReecrireCaractere(char caractere, Fichier *fichier);
114 * \brief Remet un caractère après une lecture, permet de revenir en arrière avec le pointe
115 * \param char caractere : le caractère que l'on vient de lire que l'on veut remettre
116 * \param Fichier fichier : le fichier que l'on veut modifier
117 * \author Antoine Augusti
118 */
119 void FichierTexte_deplacementCurseurMoinsUn(Fichier *fichier);
120
121 /**
122 * \fn unsigned int FichierTexte_existe(char* emplacement)
123 * \brief Détermine si un fichier existe en fonction de son emplacement
124 * \param char* emplacement : l'emplacement du fichier que l'on veut tester
125 * \return 1 si le fichier existe, 0 sinon
126 * \author Antoine Augusti
127 */
128 unsigned int FichierTexte_existe(char* emplacement);
129
130 /**
131 * \fn unsigned int FichierTexte_nombreDeCaractere(Fichier fichier);
132 * \brief Renvoi le nombre de caractère d'un Fichier
133 * \param Fichier fichier : le fichier que l'on veut étudier
134 * \return unsigned int : le nombre de caractères du fichier
135 * \author Étienne Batise
136 */
137 unsigned int FichierTexte_nombreDeCaractere(Fichier fichier);
138
139 /**
140 * \fn unsigned int FichierTexte_estUnRetourChariot(char caractere);
141 * \brief Permet de savoir si le caractere est une retour chariot
142 * \param char caractere : le caractère à étudier
143 * \return unsigned int : vrai si le caractère vaut \n
```

```
144 * \author Étienne Batise
145 */
146 unsigned int FichierTexte_estUnRetourChariot(char caractere);
147 #endif
```

## 1.5 ListeDeMot

### 1.5.1 Public

```
1  /**
2   * \file TADListeDeMot.h
3   * \brief Implantation du TAD ListeDeMot avec la SDD ListeDeMot
4   * \version 1.0
5   */
6  #ifndef __TAD_LISTE_DE_MOT__
7  #define __TAD_LISTE_DE_MOT__
8  #include "TADMot.h"
9
10 typedef struct ListeDeMot{
11     Mot* tableau;
12     unsigned int longueur;
13 } ListeDeMot;
14
15 /**
16 * \fn ListeDeMot ListeDeMot_liste();
17 * \brief Crée une Liste de Mot initialisée
18 * \return une ListeDeMot
19 * \author Étienne Batise
20 */
21 ListeDeMot ListeDeMot_liste();
22
23 /**
24 * \fn int ListeDeMot_estVide(ListeDeMot liste);
25 * \brief Permet de savoir si une Liste de Mot ne contient aucun Mot
26 * \param ListeDeMot liste : la liste à étudier
27 * \return int : vrai si elle ne contient rien, faux sinon
28 * \author Étienne Batise
29 */
30 int ListeDeMot_estVide(ListeDeMot liste);
31
32 /**
33 * \fn int ListeDeMot_longueur(ListeDeMot liste);
34 * \brief Renvoi la longueur d'une Liste de Mot
35 * \param ListeDeMot liste : la liste à étudier
36 * \return int : La longueur de la Liste de Mot
37 * \author Étienne Batise
38 */
39 unsigned int ListeDeMot_longueur(ListeDeMot liste);
40
```

```c
/**
* \fn void ListeDeMot_ajouter(ListeDeMot *liste, Mot mot);
* \brief Ajoute un Mot à la fin d'une liste de Mot
* \param ListeDeMot *liste : le pointeur de la liste à laquelle on veut ajouter un mot
* \param Mot mot : le mot que l'on veut ajouter
* \author Étienne Batise
*/
void ListeDeMot_ajouter(ListeDeMot *liste, Mot mot);

/**
* \fn Mot ListeDeMot_obtenirMot(ListeDeMot liste, int indice);
* \brief Renvoi le Mot de la liste à un indice donné
* \param ListeDeMot liste : la liste à fouiller
* \param int indice : l'indice auquel on souhaite récupérer le mot
* \return Mot : le Mot trouvé
* \author Étienne Batise
* \warning assert(1 <= indice && indice <= ListeDeMot_longueur(liste));
*/
Mot ListeDeMot_obtenir(ListeDeMot liste, unsigned int indice);

/**
* \fn void ListeDeMot_inserer(ListeDeMot *liste, Mot mot, int i);
* \brief Inserer un Mot dans la liste à un indice donné
* \param ListeDeMot liste : la liste que l'on veut modifier
* \param Mot mot : le mot que l'on veut insérer
* \param int indice : l'indice auquel on souhaite insérer le mot
* \author Étienne Batise
* \warning assert(1 <= indice && indice <= ListeDeMot_longueur(*liste) + 1);
*/
void ListeDeMot_inserer(ListeDeMot *liste, Mot mot, unsigned int i);

/**
* \fn void ListeDeMot_supprimer(ListeDeMot *liste, Mot mot, int i);
* \brief Supprime un Mot dans la liste à un indice donné
* \param ListeDeMot liste : la liste que l'on veut modifier
* \param Mot mot : le mot que l'on veut insérer
* \param int indice : l'indice auquel on souhaite supprimer le mot
* \author Étienne Batise
* \warning assert(1 <= indice && indice <= ListeDeMot_longueur(liste));
*/
void ListeDeMot_supprimer(ListeDeMot *liste, unsigned int i);

/**
* \fn char* ListeDeMot_listeDeMotEnChaine(ListeDeMot liste);
* \brief Recupère un chaine de caractère remplie des mots d'une liste séparés par des espa
* \param ListeDeMot liste : La liste à partir de laquelle on veut récupérer la chaine
* \return char* : la chaine refaite
* \warning : Le chaine est allouée dynamiquement. Il faut donc penser à la libérer après u
* \author Étienne Batise
*/
```

```
91  void ListeDeMot_listeDeMotEnChaine(ListeDeMot liste);
92
93  /**
94      TODO (Etienne)
95  */
96  void ListeDeMot_liberer(ListeDeMot *liste);
97
98  /**
99      TODO (Etienne)
100 */
101 unsigned int ListeDeMot_estPresent(ListeDeMot liste, Mot mot);
102 #endif
```

### 1.5.2 Privé

```
1  #ifndef __TAD_LISTE_DE_MOT_PRIVE__
2  #define __TAD_LISTE_DE_MOT_PRIVE__
3  #include "TADListeDeMot.h"
4  /**
5  * \fn void ListeDeMot_decalerADroite(ListeDeMot *liste, int indice);
6  * \brief Décale vers la droite tous les Mot d'une Liste de Mot à partir d'un indice. Atten
7  * \param ListeDeMot *liste :  la liste que l'on veut modifier
8  * \param int indice : l'indice à partir duquel on veut faire le décalage
9  * \author Étienne Batise
10 */
11 void ListeDeMot_decalerADroite(ListeDeMot *liste, unsigned int indice);
12
13 /**
14 * \fn void ListeDeMot_decalerAGauche(ListeDeMot *liste, int indice);
15 * \brief Décale vers la gauche tous les Mot d'une Liste de Mot à partir d'un indice. Pense
16 * \param ListeDeMot *liste :  la liste que l'on veut modifier
17 * \param int indice : l'indice à partir duquel on veut faire le décalage
18 * \author Étienne Batise
19 */
20 void ListeDeMot_decalerAGauche(ListeDeMot *liste, unsigned int indice);
21
22 #endif
```

## 1.6 ListeDeSuperMot

### 1.6.1 Public

```
1  /**
2   * \file TADListeDeSuperMot.h
3   * \brief Implantation du TAD ListeDeSuperMot avec la SDD ListeDeSuperMot
4   * \version 1.0
5   */
6  #ifndef __TAD_LISTE_DE_SUPER_MOT__
7  #define __TAD_LISTE_DE_SUPER_MOT__
8  #include "TADSuperMot.h"
```

```c

typedef struct ListeDeSuperMot{
    SuperMot* tableau;
    unsigned int longueur;
} ListeDeSuperMot;

/**
* \fn ListeDeSuperMot TADListeDeSuperMot_liste();
* \brief Crée une Liste de SuperMot initialisée
* \return une ListeDeSuperMot
* \author Étienne Batise
*/
ListeDeSuperMot ListeDeSuperMot_liste();

/**
* \fn int ListeDeSuperMot_estVide(ListeDeSuperMot liste);
* \brief Permet de savoir si une Liste de SuperMot ne contient aucun SuperMot
* \param ListeDeSuperMot liste : la liste à étudier
* \return int : vrai si elle ne contient rien, faux sinon
* \author Étienne Batise
*/
int ListeDeSuperMot_estVide(ListeDeSuperMot liste);

/**
* \fn int ListeDeSuperMot_longueur(ListeDeSuperMot liste);
* \brief Renvoi la longueur d'une Liste de SuperMot
* \param ListeDeSuperMot liste : la liste à étudier
* \return int : La longueur de la Liste de SuperMot
* \author Étienne Batise
*/
unsigned int ListeDeSuperMot_longueur(ListeDeSuperMot liste);

/**
* \fn void ListeDeSuperMot_ajouter(ListeDeSuperMot *liste, SuperMot mot);
* \brief Ajoute un Mot à la fin d'une liste de SuperMot
* \param ListeDeSuperMot *liste : le pointeur de la liste à laquelle on veut ajouter un Su
* \param SuperMot superMot : le SuperMot que l'on veut ajouter
* \author Étienne Batise
*/
void ListeDeSuperMot_ajouter(ListeDeSuperMot *liste, SuperMot superMot);

/**
* \fn Mot ListeDeSuperMot_obtenirMot(ListeDeSuperMot liste, int indice);
* \brief Renvoi le SuperMot de la liste à un indice donné
* \param ListeDeSuperMot liste : la liste à fouiller
* \param int indice : l'indice auquel on souhaite récupérer le SuperMot
* \return SuperMot : le SuperMot trouvé
* \author Étienne Batise
* \warning assert(1 <= indice && indice <= ListeDeSuperMot_longueur(liste));
*/
```

```
59 SuperMot ListeDeSuperMot_obtenir(ListeDeSuperMot liste,unsigned int indice);
60
61 /**
62 * \fn void ListeDeSuperMot_inserer(ListeDeSuperMot *liste, SuperMot superMot, int i);
63 * \brief Inserer un SuperMot dans la liste à un indice donné
64 * \param ListeDeSuperMot liste : la liste que l'on veut modifier
65 * \param SuperMot superMot : le Supermot que l'on veut inserer
66 * \param int indice : l'indice auquel on souhaite inserer le Supermot
67 * \author Étienne Batise
68 * \warning assert(1 <= indice && indice <= ListeDeSuperMot_longueur(*liste) + 1);
69 */
70 void ListeDeSuperMot_inserer(ListeDeSuperMot *liste, SuperMot superMot, unsigned int i);
71
72 /**
73 * \fn void ListeDeSuperMot_supprimer(ListeDeSuperMot *liste, SueprMot superMot, int i);
74 * \brief Supprime un SuperMot dans la liste à un indice donné
75 * \param ListeDeSuperMot liste : la liste que l'on veut modifier
76 * \param SuperMot superMot : le SuperMot que l'on veut inserer
77 * \param int indice : l'indice auquel on souhaite supprimer le Supermot
78 * \author Étienne Batise
79 * \warning assert(1 <= indice && indice <= ListeDeSuperMot_longueur(liste));
80 */
81 void ListeDeSuperMot_supprimer(ListeDeSuperMot *liste, unsigned int i);
82
83 /**
84     TODO (Etienne)
85 */
86 void ListeDeSuperMot_liberer(ListeDeSuperMot *liste);
87 #endif
```

### 1.6.2 Privé

```
1  #ifndef __TAD_LISTE_DE_SUPER_MOT_PRIVE__
2  #define __TAD_LISTE_DE_SUPER_MOT_PRIVE__
3  #include "TADListeDeSuperMot.h"
4  /**
5  * \fn void ListeDeSuperMot_decalerADroite(ListeDeSuperMot *liste, int indice);
6  * \brief Décale vers la droite tous les Mot d'une Liste de SuperMot à partir d'un indice.
7  * \param ListeDeSuperMot *liste :  la liste que l'on veut modifier
8  * \param int indice : l'indice à partir duquel on veut faire le décalage
9  * \author Étienne Batise
10 */
11 void ListeDeSuperMot_decalerADroite(ListeDeSuperMot *liste, unsigned int indice);
12
13 /**
14 * \fn void ListeDeSuperMot_decalerAGauche(ListeDeSuperMot *liste, int indice);
15 * \brief Décale vers la gauche tous les Mot d'une Liste de SuperMot à partir d'un indice.
16 * \param ListeDeSuperMot *liste :  la liste que l'on veut modifier
17 * \param int indice : l'indice à partir duquel on veut faire le décalage
18 * \author Étienne Batise
```

```
19 */
20 void ListeDeSuperMot_decalerAGauche(ListeDeSuperMot *liste, unsigned int indice);
21
22 #endif
```

## 1.7 Mot

### 1.7.1 Public

```
1  /**
2   * \file TADMot.h
3   * \brief Implantation du TAD Mot avec la SDD Mot
4   * \version 1.0
5   */
6  #ifndef __TAD_MOT__
7  #define __TAD_MOT__
8
9  typedef struct {
10   char* contenu;
11   unsigned int longueur;
12 } Mot;
13
14 /**
15  * \fn Mot Mot_mot();
16  * \brief Crée un Mot vide
17  * \return retourne un Mot "vide"
18  * \author Thibaud Dauce
19  */
20 Mot Mot_mot();
21
22 /**
23  * \fn char* Mot_obtenirContenu(Mot mot);
24  * \brief Retourne le contenu (les lettres) d'un Mot
25  * \param Mot mot : le mot concerné
26  * \return Le contenu du mot
27  * \author Thibaud Dauce
28  */
29 char* Mot_obtenirContenu(Mot mot);
30
31 /**
32  * \fn unsigned int Mot_longueur(Mot mot);
33  * \brief Retourne la longueur d'un mot
34  * \param Mot mot : le mot concernÃ©
35  * \return la taille du mot
36  * \author Thibaud Dauce
37  */
38 unsigned int Mot_longueur(Mot mot);
39
40 /**
```

```
41  * \fn  Mot Mot_genererMot(char * chaine);
42  * \brief Génère un mot à partir d'une chaine de caractère
43  * \param char* chaine : la chaine à partir de laquelle on veut obtenir un mot
44  * \return le mot à partir de la chaine
45  * \author Thibaud Dauce
46  */
47 Mot Mot_genererMot(char* chaine);
48
49 /**
50  * \fn unsigned int Mot_sontEgaux(Mot mot1, Mot mot2);
51  * \brief Indique si 2 Mots sont égaux (même contenu, de même taille)
52  * \param Mot mot1 : le premier mot concerné
53  * \param Mot mot2 : le deuxième mot concerné
54  * \return vrai si les mots sont égaux, faux sinon
55  * \author Thibaud Dauce
56  */
57 unsigned int Mot_sontEgaux(Mot mot1, Mot mot2);
58
59 /**
60  * \fn void Mot_insererCaractere(Mot* mot, char c, unsigned int position);
61  * \brief Insère un caractère dans un Mot à la position donnée
62  * \param Mot mot : le mot concerné
63  * \param char c : le caractère concerné
64  * \param unsigned int position : la position
65  * \author Thibaud Dauce
66  */
67 void Mot_insererCaractere(Mot* mot, char c, unsigned int position);
68
69
70 /**
71  * \fn void Mot_modifierCaractere(Mot* mot, char c, unsigned int position);
72  * \brief Modifie un caractère dans un Mot à la position donnée
73  * \param Mot mot : le mot concerné
74  * \param char c : le caractère concerné
75  * \param unsigned int position : la position
76  * \author Thibaud Dauce
77  */
78 void Mot_modifierCaractere(Mot* mot, char c, unsigned int position);
79
80 /**
81  * \fn void Mot_supprimerCaractere(Mot* mot, unsigned int position);
82  * \brief Supprime un caractère dans un Mot à la position donnée
83  * \param Mot mot : le mot concerné
84  * \param unsigned int position : la position
85  * \author Thibaud Dauce
86  */
87 void Mot_supprimerCaractere(Mot* mot, unsigned int position);
88
89 /**
90  * \fn char Mot_iemeCaracte(Mot mot, unsigned int position);
```

```
91   * \brief Retourne le caractère d'un Mot à la position donnée
92   * \param Mot mot : le mot concerné
93   * \param unsigned int position : la position
94   * \return Le caractère en question
95   * \author Thibaud Dauce
96   */
97  char Mot_iemeCaractere(Mot mot, unsigned int position);
98
99  /**
100  * \fn Mot Mot_concatener(Mot mot1, Mot mot2);
101  * \brief Concatène deux mots
102  * \param Mot mot1 : le premier mot concerné
103  * \param Mot mot2 : le deuxième mot concerné
104  * \return Le Mot résultat de la concaténation des deux mots
105  * \author Thibaud Dauce
106  */
107 Mot Mot_concatener(Mot mot1, Mot mot2);
108
109 /**
110  * \fn unsigned int Mot_estUnMotValide(Mot mot);
111  * \brief Indique si un Mot est bien composé de lettres ou de tirets
112  * \param Mot mot : le mot concerné
113  * \return vrai si le mot est valide, faux sinon
114  * \author Thibaud Dauce
115  */
116 unsigned int Mot_estUnMotValide(Mot mot);
117
118 /**
119  * \fn void Mot_separerMots(Mot mot, unsigned int position, Mot* mot1, Mot* mot2);
120  * \brief Sépare en deux mots. L'indice donné est le début du deuxième mot. separerMots(ab
121  * \param Mot mot : le mot concerné
122  * \param unsigned int position : la position à laquelle le deuxième mot doit commencer
123  * \param Mot mot1 : le premier sous-mot
124  * \param Mot mot2 : le deuxième sous-mot
125  * \author Thibaud Dauce
126  */
127 void Mot_separerMots(Mot mot, unsigned int position, Mot* mot1, Mot* mot2);
128
129 /**
130  * \fn Mot Mot_sousMot(Mot mot, unsigned int debut, unsigned int longueur);
131  * \brief Crée un sous-mot à partir de l'indice 'debut' du mot et d'une longueur 'longueur
132  * \param Mot mot : le mot concerné
133  * \param unsigned int position : la position à laquelle le sous-mot doit commencer
134  * \param unsigned int longueur : la longueur du sous-mot à partir de l'indice de début
135  * \return le sous-mot créé
136  * \author Thibaud Dauce
137  */
138 Mot Mot_sousMot(Mot mot, unsigned int debut, unsigned int longueur);
139
140 /**
```

```
 * \fn Mot Mot_sousMot(Mot mot, unsigned int debut, unsigned int longueur);
 * \brief Indique si le Mot mot1 est contenu dans le Mot mot2
 * \param Mot mot1 : le mot recherché
 * \param Mot mot2 : le mot où l'on recherche
 * \return vrai si mot1 est un sous-mot de mot2, faux sinon
 * \author Thibaud Dauce
 */
unsigned int Mot_estSousMot(Mot mot1, Mot mot2);

/**
 * \fn unsigned int Mot_sontIdentiques(Mot* mot1, Mot* mot2);
 * \brief Indique si le Mot mot1 à la même adresse que le Mot mot2
 * \param Mot mot1 : le premier mot
 * \param Mot mot2 : le deuxième mot
 * \return vrai si mot1 à la même adresse que mot2, faux sinon
 * \author Thibaud Dauce
 */
unsigned int Mot_sontIdentiques(Mot* mot1, Mot* mot2);

#endif
```

### 1.7.2 Privé

```
#ifndef __TAD_MOT_PRIVE__
#define __TAD_MOT_PRIVE__
#include "TADMot.h"
/**
* \fn unsigned int max(unsigned int a, unsigned int b);
* \brief Retour le maximum entre deux int
* \param int a : premier integer
* \param int b : deuxième integer
* \return le plus grand des deux
* \author Thibaud Dauce
*/
unsigned int max(unsigned int a, unsigned int b);

/**
* \fn unsigned int estLettreOuTiret(char c);
* \brief Test d'un caractère
* \param char c : le caractère a tester
* \return vrai si le caractère passé en paramètre est valide
* \author Thibaud Dauce
*/
unsigned int estLettreOuTiret(char c);

#endif
```

## 1.8 Parametres

```c
/**
 * \file TADParametres.h
 * \brief Implantation du TAD Parametres avec la SDD Parametres
 * \version 1.0
 */
#ifndef __TAD_PARAMETRES__
#define __TAD_PARAMETRES__
typedef struct {
  char *dictionnaire;
  char *fichier;
  char *cible;
  int     aide;
} Parametres;

/**
 * \fn Parametres Parametres_parametre();
 * \brief Crée un Parametre
 * \return retourne le Parametre créé
 * \author Faustine Demiselle
 */
Parametres Parametres_parametre();

/**
 * \fn void Parametres_fixerDictionnaire(Parametres* p, char* nomDico);
 * \brief Rempli le champ dictionnaire du paramètre p
 * \param Parametres* p : le paramètre que l'on veut modifier
 * \param  char* nomDico : le nom du dictionnaire
 * \author Faustine Demiselle
 */
void Parametres_fixerDictionnaire(Parametres* p, char* nomDuDico);

/**
 * \fn void Parametres_fixerFichier(Parametres* p, char* nomDuFichier);
 * \brief Rempli le champ fichier du paramètre p
 * \param Parametres* p : le paramètre que l'on veut modifier
 * \param  char* nomDuFichier : le nom du fichier
 * \author Faustine Demiselle
 */
void Parametres_fixerFichier(Parametres* p, char* nomDuFichier);

/**
 * \fn void Parametres_fixerCible(Parametres* p, char* nomDeLaCible);
 * \brief Rempli le champ cible du paramètre p
 * \param Parametres* p : le paramètre que l'on veut modifier
 * \param  char* nomDeLaCible : le nom de la cible
 * \author Faustine Demiselle
 */
void Parametres_fixerCible(Parametres* p, char* nomDeLaCible);
```

```
49
50  /**
51   * \fn void Parametres_fixerAide(Parametres* p, int presenceDuH);
52   * \brief Permet de fixer l'aide comme paramètre si "-h" est utilisé
53   * \param Parametres* p : le paramètre que l'on veut modifier
54   * \param int presenceDuH : vrai(1) ou faux(0) selon que l'on est trouvé le "-h"
55   * \author Faustine Demiselle
56   */
57  void Parametres_fixerAide(Parametres* p, int presenceDuH);
58
59  /**
60   * \fn char* Parametres_obtenirDictionnaire(Parametres p);
61   * \brief Retourne le nom du dictionnaire
62   * \param Parametres p : le paramètre dont on veut obtenir le nom du dico
63   * \return le nom du dictionnaire
64   * \author Faustine Demiselle
65   */
66  char* Parametres_obtenirDictionnaire(Parametres p);
67
68  /**
69   * \fn char* Parametres_obtenirFichier(Parametres p);
70   * \brief Retourne le nom du fichier
71   * \param Parametres p : le paramètre dont on veut obtenir le nom du fichier
72   * \return le nom du fichier
73   * \author Faustine Demiselle
74   */
75  char* Parametres_obtenirFichier(Parametres p);
76
77  /**
78   * \fn char* Parametres_obtenirCible(Parametres p);
79   * \brief Retourne le nom de la cible
80   * \param Parametres p : le paramètre dont on veut obtenir le nom de la cible
81   * \return le nom du cible
82   * \author Faustine Demiselle
83   */
84  char* Parametres_obtenirCible(Parametres p);
85
86  /**
87   * \fn char* Parametres_obtenirAide(Parametres p);
88   * \brief Retourne la présence de l'aide
89   * \param Parametres p : le paramètre dont on veut obtenir la présence de l'aide
90   * \return la présence de l'aide
91   * \author Faustine Demiselle
92   */
93  int Parametres_obtenirAide(Parametres p);
94  #endif
```

## 1.9 SuperMot

```c
/**
 * \file TADSuperMot.h
 * \brief Implantation du TAD SuperMot avec la SDD SuperMot
 * \version 1.0
 */
#ifndef __TAD_SUPER_MOT__
#define __TAD_SUPER_MOT__
#include "TADMot.h"
#include "TADListeDeMot.h"

typedef struct {
    Mot mot;
    unsigned int position;
    ListeDeMot listeDeCorrections;
    int valide;
} SuperMot;

/**
* \fn SuperMot_superMot
* \brief initialise un superMot vide
* \return le superMot vide
* \author Faustine Demiselle
*/
SuperMot SuperMot_superMot();

/**
* \fn SuperMot_obtenirMot
* \brief obtenir le mot du superMot
* \param SuperMot superMot : le superMot dont on veut le mot
* \return le mot du superMot
* \author Faustine Demiselle
*/
Mot SuperMot_obtenirMot(SuperMot superMot);

/**
* \fn SuperMot_fixerMot
* \brief fixe mot dans superMot
* \param SuperMot superMot : le superMot à modifier
* \param Mot mot : le mot à fixer
* \return void
* \author Faustine Demiselle
*/
void SuperMot_fixerMot(SuperMot* superMot, Mot mot);

/**
* \fn SuperMot_obtenirPosition
* \brief obtenir la position affectée au superMot
* \param SuperMot superMot : le superMot dont on cherche position
```

```
49  * \return la position
50  * \author Faustine Demiselle
51  */
52  unsigned int SuperMot_obtenirPosition(SuperMot superMot);
53
54  /**
55  * \fn SuperMot_fixerposition
56  * \brief fixe la position du superMot
57  * \param SuperMot superMot : le superMot à modifier
58  * \param unsigned int position : la position à fixer
59  * \return void
60  * \author Faustine Demiselle
61  */
62  void SuperMot_fixerPosition(SuperMot* superMot, unsigned int position);
63
64  /**
65  * \fn SuperMot_obtenirListeDeCorrection
66  * \brief Obtenir la liste de corrections (listeDeMot) affectée à un superMot
67  * \param SuperMot superMot : le superMot concerné
68  * \return ListeDeMot
69  * \author Faustine Demiselle
70  */
71  ListeDeMot SuperMot_obtenirListeDeCorrection(SuperMot superMot);
72
73  /**
74  * \fn SuperMot_fixerListeDeCorrections
75  * \brief fixe les corrections (listeDeMot) dans superMot
76  * \param SuperMot superMot : le superMot à modifier
77  * \param ListeDeMot corrections : les corrections à fixer
78  * \return void
79  * \author Faustine Demiselle
80  */
81  void SuperMot_fixerListeDeCorrections(SuperMot* superMot, ListeDeMot corrections);
82
83  /**
84  * \fn SuperMot_obtenirValidite
85  * \brief obtien la validité du superMot
86  * \param SuperMot superMot : le superMot concerné
87  * \return int (à lire comme un booléen)
88  * \author Faustine Demiselle
89  */
90  int SuperMot_obtenirValidite(SuperMot superMot);
91
92  /**
93  * \fn SuperMot_fixerValidite
94  * \brief fixe la validité du superMot
95  * \param SuperMot superMot : le superMot à modifier
96  * \param int estValide : la valeur (validité) à fixer
97  * \return void
98  * \author Faustine Demiselle
```

```
99  */
100 void SuperMot_fixerValidite(SuperMot* superMot, int estValide);
101
102 #endif
```

# Chapitre 2

# Fichiers headers de la logique métier

## Sommaire



## 2.1 affichage

```
1  /**
2   * \file affichage.h
3   * \brief Fonctions utiles pour l'affichage
4   * \version 1.0
5   */
6
7  #ifndef __IHM__
8  #define __IHM__
9
10 /**
11 * \fn void IHM_afficherAide();
12 * \brief Affiche l'aide du programme
13 * \author Étienne Batise
14 */
15 void IHM_afficherAide();
16
17 /**
18 * \fn  void IHM_afficherResultat(int erreur, char* resultat);
19 * \brief Affiche le résultat du programme
20 * \param int erreur : le type d'erreur (0 =  aucune erreur)
21 * \param char* resultat : la chaine à écrire si il n'y a aucune erreur
22 * \author Étienne Batise
23 */
24 void IHM_afficherResultat(int erreur, char* resultat);
25
26 /**
27 * \fn void IHM_afficherCorrection(char* resultat);
28 * \brief Affiche la correction d'une chaine
```

```
29 * \param char* chaine : la chaine à afficher
30 * \author Étienne Batise
31 */
32 void IHM_afficherCorrection(char* resultat);
33
34 /**
35 * \fn void IHM_afficherErreur(int erreur);
36 * \brief Affiche le type d'erreur rencontrée
37 * \param int erreur : l'erreur rencontrée ( > 0 )
38 * \author Étienne Batise
39 */
40 void IHM_afficherErreur(int erreur);
41
42 #endif
```

## 2.2 corriger

```
1  /**
2   * \file corriger.h
3   * \brief Fonctions de haut-niveau utiles pour la correction
4   * \version 1.0
5   */
6  #ifndef __CORRIGER__
7  #define __CORRIGER__
8  #include "TADDictionnaire.h"
9  #include "TADListeDeSuperMot.h"
10
11 /**
12 * \fn char* corriger(char* chaine, Dictionnaire dictionnaire);
13 * \brief Affiche une correction pour une chaine de caractères selon un dictionnaire
14 * \param char* chaine : la chaine à corriger
15 * \param Dictionnaire dictionnaire : le dictionnaire
16 * \author
17 */
18 void corriger(char* chaine, Dictionnaire dictionnaire);
19
20 /**
21 * \fn void evaluerSaisie(unsigned int* estValide, int* erreur, char* chaine);
22 * \brief Etudie une chaine de caractères selon plusieurs critères
23 * \param unsigned int* estValide : la validité de la chaine de caractères
24 * \param int* erreur : le type d'erreur. N'a pas de sens si estValide vaut 1
25 * \param char* chaine :la chaine de caractère à étudier
26 */
27 void evaluerSaisie(unsigned int* estValide, unsigned int* erreur, char* chaine);
28 /**
29 * \fn char* lireChaine()
30 * \brief Lit une chaîne de caractères rentrée dans l'entrée standard. Celle-ci DOIT se ter
31 * \author Antoine Augusti
32 * \warning L'allocation est faites dans la fonction. La désallocation est À FAIRE
```

```
33 */
34 char* lireChaine();
35
36 /**
37 * \fn void corrigerListeDeSuperMots(ListeDeSuperMot listeDeSuperMots, Dictionnaire diction
38 * \brief Corrige une liste de SuperMot selon un dictionnaire
39 * \param ListeDeSuperMot* listeDeSuperMots : la liste de SupeMot à corriger
40 * \param Dictionnaire dictionnaire : le dictionnaire
41 */
42 void corrigerListeDeSuperMots(ListeDeSuperMot* listeDeSuperMots, Dictionnaire dictionnaire);
43
44 /**
45 * \fn void afficherCorrection(ListeDeSuperMot listeDeSuperMots);
46 * \brief Crée une chaine de caractères qui sera affichée
47 * \param ListeDeSuperMot listeDeSuperMots : la liste de SuperMot qui permet de construire
48 */
49 void afficherCorrection(ListeDeSuperMot listeDeSuperMots);
50
51 #endif
```

## 2.3 transtypage

```
1  /**
2   * \file transtypage.h
3   * \brief Module permettant d'effectuer plusieurs types de transtypages
4   * \version 1.0
5   */
6  #ifndef __TRANSTYPAGE__
7  #define __TRANSTYPAGE__
8
9  /**
10 * \fn char* entierEnChaine(int e)
11 * \brief Permet de récupérer un Naturel sous la forme d'une chaine de caractère
12 * \param unsigned int e : l'entier que l'on veut transformer
13 * \return char* : la chaine de caractère correspondant à l'entier
14 * \warning : Le chaine est allouée dynamiquement. Il faut donc penser à la libérer après u
15 * \author Étienne Batise
16 */
17 char* Transtypage_entierEnChaine(unsigned int e);
18 #endif
```

# Chapitre 3

# Fichiers sources des TAD

## Sommaire



## 3.1 CorrecteurOrthographique

```c
#include <stdio.h>
#include <stdlib.h>
#include "TADCorrecteurOrthographique.h"
#include "TADCorrecteurOrthographiquePrive.h"
#include "TADMot.h"
#include "TADDictionnaire.h"
#include "TADListeDeMot.h"
#define NB_LETTRES_ALPHABET 26

/**
* \see TADCorrecteurOrthographique.h
*/
ListeDeMot CorrecteurOrthographique_proposerMots(Dictionnaire dictionnaire, Mot mot) {
    return CorrecteurOrthographique_motsExacts(CorrecteurOrthographique_possibilitesMots(mot
}

/**
* \see TADCorrecteurOrthographiquePrive.h
*/
ListeDeMot CorrecteurOrthographique_possibilitesMots(Mot mot) {
    /* Déclarations */
    ListeDeMot liste, liste2, liste3, liste4;
    unsigned int i;
```

```
24
25      /* Corps */
26      liste = CorrecteurOrthographique_genererMotsParSuppression(mot);
27      liste2 = CorrecteurOrthographique_genererMotsParRemplacement(mot);
28      liste3 = CorrecteurOrthographique_genererMotsParInsertion(mot);
29      liste4 = CorrecteurOrthographique_genererMotsParInversion(mot);
30
31      for (i = 1; i <= ListeDeMot_longueur(liste2); i++) {
32          ListeDeMot_ajouter(&liste, ListeDeMot_obtenir(liste2, i));
33      }
34      for (i = 1; i <= ListeDeMot_longueur(liste3); i++) {
35          ListeDeMot_ajouter(&liste, ListeDeMot_obtenir(liste3, i));
36      }
37      for (i = 1; i <= ListeDeMot_longueur(liste4); i++) {
38          ListeDeMot_ajouter(&liste, ListeDeMot_obtenir(liste4, i));
39      }
40
41      return liste;
42  }
43
44  ListeDeMot CorrecteurOrthographique_genererMotsParInversion(Mot mot) {
45      /* Déclarations */
46      ListeDeMot listeDeMots;
47      Mot motTemp;
48      char c;
49      unsigned int i;
50
51      /* Corps */
52      listeDeMots = ListeDeMot_liste();
53      for (i = 1; i <= Mot_longueur(mot) - 1; i++) {
54          motTemp = Mot_genererMot(Mot_obtenirContenu(mot));
55          c = Mot_iemeCaractere(mot, i);
56          Mot_modifierCaractere(&motTemp, Mot_iemeCaractere(mot, i + 1), i);
57          Mot_modifierCaractere(&motTemp, c, i + 1);
58          ListeDeMot_inserer(&listeDeMots, motTemp, 1);
59      }
60      return listeDeMots;
61  }
62
63  /**
64  * \see TADCorrecteurOrthographiquePrive.h
65  */
66  ListeDeMot CorrecteurOrthographique_genererMotsParSuppression(Mot mot) {
67      /* Déclarations */
68      ListeDeMot listeDeMots;
69      Mot motTemp;
70      unsigned int i;
71
72      /* Corps */
73      listeDeMots = ListeDeMot_liste();
```

```c
    for (i = 1; i <= Mot_longueur(mot); i++) {
        motTemp = Mot_genererMot(Mot_obtenirContenu(mot));
        Mot_supprimerCaractere(&motTemp, i);
        ListeDeMot_inserer(&listeDeMots, motTemp, 1);
    }
    return listeDeMots;
}

/**
* \see TADCorrecteurOrthographiquePrive.h
*/
ListeDeMot CorrecteurOrthographique_genererMotsParRemplacement(Mot mot) {
    /* Déclarations */
    char lettre[NB_LETTRES_ALPHABET];
    ListeDeMot listeDeMots;
    Mot motTemp;
    unsigned int i, j;

    /* Corps */
    listeDeMots = ListeDeMot_liste();
    lettre[0] = 'a';

    /* Initiatilisation du tableau de lettres */
    for (i = 1; i < NB_LETTRES_ALPHABET; i++) {
        lettre[i] = lettre[i - 1] + 1;
    }

    for (i = 1; i <= Mot_longueur(mot); i++) {
        for (j = 0; j < NB_LETTRES_ALPHABET; j++) {
            motTemp = Mot_genererMot(Mot_obtenirContenu(mot));
            Mot_modifierCaractere(&motTemp, lettre[j], i);
            ListeDeMot_inserer(&listeDeMots, motTemp, 1);
        }
    }

    return listeDeMots;
}

/**
* \see TADCorrecteurOrthographiquePrive.h
*/
ListeDeMot CorrecteurOrthographique_genererMotsParInsertion(Mot mot) {
    /* Déclarations */
    char lettre[NB_LETTRES_ALPHABET];
    ListeDeMot listeDeMots;
    Mot motTemp;
    unsigned int i, j;

    /* Corps */
    listeDeMots = ListeDeMot_liste();
```

```
124     lettre[0] = 'a';
125 
126     /* Initiatilisation du tableau de lettres */
127     for (i = 1; i < NB_LETTRES_ALPHABET; i++) {
128         lettre[i] = lettre[i - 1] + 1;
129     }
130 
131     for (i = 1; i <= Mot_longueur(mot) + 1 ; i++) {
132         for (j = 0; j < NB_LETTRES_ALPHABET; j++) {
133             motTemp = Mot_genererMot(Mot_obtenirContenu(mot));
134             Mot_insererCaractere(&motTemp, lettre[j], i);
135             ListeDeMot_inserer(&listeDeMots, motTemp, 1);
136         }
137     }
138 
139     return listeDeMots;
140 
141 }
142 
143 /**
144 * \see TADCorrecteurOrthographiquePrive.h
145 */
146 ListeDeMot CorrecteurOrthographique_motsExacts(ListeDeMot listeDeMots, Dictionnaire dictionna
147     /* Déclarations */
148     Mot mot;
149     ListeDeMot listeFinale;
150     unsigned int i;
151 
152     /* Corps */
153     listeFinale = ListeDeMot_liste();
154     for (i = 1; i <= ListeDeMot_longueur(listeDeMots); i++) {
155         mot = ListeDeMot_obtenir(listeDeMots, i);
156         if (Dictionnaire_estPresent(dictionnaire, mot) && !ListeDeMot_estPresent(listeFinale
157             ListeDeMot_ajouter(&listeFinale, mot);
158         }
159     }
160 
161     return listeFinale;
162 }
```

## 3.2 Dictionnaire

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <ctype.h>
4 #include <assert.h>
5 #include <string.h>
6 #include "TADDictionnaire.h"
7 #include "TADElementDictionnairePrive.h"
```

```c
#include "TADMot.h"
#include "TADFichierTexte.h"

/* Partie privée */

/**
* \see TADElementDictionnairePrive.h
*/
ElementDictionnaire ElementDico_element(char caractere, unsigned int estPresent) {
    unsigned int i;
    NoeudDictionnaire *noeud = (NoeudDictionnaire *)malloc(sizeof(NoeudDictionnaire));

    noeud->lettre = caractere;
    noeud->presence = estPresent;
    noeud->longueur = 0;
    for (i = 0; i < NB_LETTRES; ++i) {
        noeud->tableauDesElementsSuivants[i] = NULL;
    }

    return noeud;
}

/**
* \see TADElementDictionnairePrive.h
*/
void ElementDico_ajouterElementSuivant(ElementDictionnaire *e1, ElementDictionnaire e2) {
    unsigned int indice;

    indice = (*e1)->longueur;
    (*e1)->longueur = (*e1)->longueur + 1;
    (*e1)->tableauDesElementsSuivants[indice] = e2;
}

/**
* \see TADElementDictionnairePrive.h
*/
void ElementDico_supprimerElementSuivant(ElementDictionnaire *e1, unsigned int indice) {
    unsigned int i, j;

    assert(0 <= indice && indice < ElementDico_obtenirLongueurTableauDesElementsSuivants(*e1

    /* On supprime tous les fils de l'élément que l'on veut supprimer */
    for (j = 0; j < ElementDico_obtenirLongueurTableauDesElementsSuivants((*e1)->tableauDesE
        ElementDico_supprimerElementSuivant(&((*e1)->tableauDesElementsSuivants[indice]), j)
    }

    /* On supprime l'élément que l'on veut supprimer et on décrémente la longueur du père
    free((*e1)->tableauDesElementsSuivants[indice]);

    (*e1)->longueur = (*e1)->longueur - 1;
```

```
58
59      /* On supprime le pointeur (qui ne pointe plus vers rien) du tableau du père en décalan
60      for (i = indice; i < (*e1)->longueur; ++i) {
61          (*e1)->tableauDesElementsSuivants[i] = (*e1)->tableauDesElementsSuivants[i + 1];
62      }
63  }
64
65  /**
66  * \see TADElementDictionnairePrive.h
67  */
68  void ElementDico_ajouterPresence(ElementDictionnaire *e1, unsigned int presence) {
69      (*e1)->presence = presence;
70  }
71
72  /**
73  * \see TADElementDictionnairePrive.h
74  */
75  unsigned int ElementDico_obtenirLongueurTableauDesElementsSuivants(ElementDictionnaire eleme
76      return element->longueur;
77  }
78
79  /**
80  * \see TADElementDictionnairePrive.h
81  */
82  ElementDictionnaire (*ElementDico_obtenirTableauDesElementsSuivants(ElementDictionnaire eleme
83      return &(element->tableauDesElementsSuivants);
84  }
85
86  /**
87  * \see TADElementDictionnairePrive.h
88  */
89  char ElementDico_obtenirLettre(ElementDictionnaire element) {
90      assert(element != NULL);
91
92      return element->lettre;
93  }
94
95  /**
96  * \see TADElementDictionnairePrive.h
97  */
98  unsigned int ElementDico_obtenirBooleen(ElementDictionnaire element) {
99      assert(element != NULL);
100
101     return element->presence;
102 }
103
104 /**
105  * \fn unsigned int Dictionnaire_recupererUneLongueur(Fichier fichier)
106  * \author Antoine Augusti
107  * \brief Récupère la longueur du tableau des élements suivants dans un fichier où l'on sa
```

```
108  * \param Fichier fichier : le fichier contenant le dictionnaire
109  * \param ElementDictionnaire e2 : l'ElementDictionnaire à ajouter
110  * \warning Fonction privée
111  */
112 unsigned int Dictionnaire_recupererUneLongueur(Fichier fichier) {
113     char premierCaractere, DeuxiemeCaractere;
114     unsigned int resultat;
115 
116     premierCaractere = FichierTexte_lireCaractere(fichier);
117 
118     /* Conversion char -> int d'après http://stackoverflow.com/questions/781668/char-to-in
119     resultat = premierCaractere - '0';
120 
121     if (!FichierTexte_finFichier(fichier)) {
122 
123             DeuxiemeCaractere = FichierTexte_lireCaractere(fichier);
124             /* On a une longueur sur 2 caractères */
125             if (isdigit(DeuxiemeCaractere))
126                 resultat = resultat * 10 + (DeuxiemeCaractere - '0');
127                 /* Longueur sur un seul caractère, on remet la lettre dans le flux */
128             else {
129                 FichierTexte_deplacementCurseurMoinsUn(&fichier);
130             }
131     }
132     return resultat;
133 }
134 
135 /**
136  * \fn ElementDictionnaire Dictionnaire_chargerElementsSuivants(Fichier fichier)
137  * \author Antoine Augusti
138  * \brief Charge récursivement les élements suivants d'un dictionnaire à partir d'un fichi
139  * \param Fichier fichier : le fichier contenant le dictionnaire
140  * \param ElementDictionnaire e2 : l'ElementDictionnaire à ajouter
141  * \warning Fonction privée
142  */
143 ElementDictionnaire Dictionnaire_chargerElementsSuivants(Fichier fichier) {
144     char caractere;
145     unsigned int booleen, i, longueur;
146     ElementDictionnaire element;
147 
148     caractere = FichierTexte_lireCaractere(fichier);
149     /* Conversion char -> int d'après http://stackoverflow.com/questions/781668/char-to-in
150     /* Il faut vérifier que l'on peut bien convertir pour avoir un int cohérent */
151     booleen = FichierTexte_lireCaractere(fichier) - '0';
152     longueur = Dictionnaire_recupererUneLongueur(fichier);
153     element = ElementDico_element(caractere, booleen);
154 
155     for (i = 1; i <= longueur; ++i) {
156         ElementDico_ajouterElementSuivant(&element, Dictionnaire_chargerElementsSuivants(fic
157     }
```

```
158
159     return element;
160 } /* Antoine */
161
162 /**
163  * \fn char Dictionnaire_booleenEnCaractere(unsigned int booleen)
164  * \author Jean-Claude Bernard
165  * \brief Transforme un booléen(unsigned int) en un caractère
166  * \param unsigned int booleen : un booleen
167  * \warning Fonction privée
168  */
169  char Dictionnaire_booleenEnCaractere(unsigned int booleen) {
170      /* Conversion d'un int en char ;  d'après http://stackoverflow.com/questions/1114741/
171      char resultat = (char)(((int)'0') + booleen);
172
173      return resultat;
174  }
175
176 /**
177  * \fn  void Dictionnaire_naturelEnCaractere(unsigned int longueur, char* dizaine, char* u
178  * \author Jean-Claude Bernard
179  * \brief Transforme un naturel en un caractère
180  * \param unsigned int longueur : le naturel que l'on veut convertir
181  * \param char* dizaine : le caractère (chiffre des dizaines) que l'on veut obtenir
182  * \param char* unite : le caractère (unité) que l'on veut obtenir
183  * \warning Fonction privée
184  */
185  void Dictionnaire_naturelEnCaractere(unsigned int longueur, char* dizaine, char* unite) {
186      assert((0 <= longueur) && (longueur <= NB_LETTRES));
187
188      *dizaine = '0';
189
190      if (longueur >= 10)
191      {
192          if (longueur >= 90)
193              *dizaine = '9';
194          else if (longueur >= 80)
195              *dizaine = '8';
196          else if (longueur >= 70)
197              *dizaine = '7';
198          else if (longueur >= 60)
199              *dizaine = '6';
200          else if (longueur >= 50)
201              *dizaine = '5';
202          else if (longueur >= 40)
203              *dizaine = '4';
204          else if (longueur >= 30)
205              *dizaine = '3';
206          else if (longueur >= 20)
207              *dizaine = '2';
```

```
208             else
209                 *dizaine = '1';
210         }
211         *unite = (char)(((int)'0') + (longueur % 10));
212     }
213
214     /**
215      * \fn void Dictionnaire_sauvegarderElementsSuivants(Fichier* fichier, ElementDictionnaire
216      * \author Jean-Claude Bernard
217      * \brief Sauvegarde récursivement les élements suivants d'un dictionnaire dans un fichier
218      * \param Fichier* fichier : le fichier dans lequel on veut sauvegarder le dictionnaire
219      * \param ElementDictionnaire elementDico : l'ElementDictionnaire à sauvegarder
220      * \warning Fonction privée
221      */
222     void Dictionnaire_sauvegarderElementsSuivants(Fichier* fichier, ElementDictionnaire elementD
223         ElementDictionnaire tableauDesElementsSuivants[NB_LETTRES];
224         char dizaine, unite;
225         unsigned int i, indice;
226
227         for (indice = 0; indice < NB_LETTRES; ++indice) {
228             tableauDesElementsSuivants[indice] = (*ElementDico_obtenirTableauDesElementsSuivants
229         }
230
231         for (i = 0; i < ElementDico_obtenirLongueurTableauDesElementsSuivants(elementDico); ++i)
232         {
233             FichierTexte_ecrireCaractere(fichier, ElementDico_obtenirLettre(tableauDesElementsSu
234             FichierTexte_ecrireCaractere(fichier, Dictionnaire_booleenEnCaractere(ElementDico_ob
235             Dictionnaire_naturelEnCaractere(ElementDico_obtenirLongueurTableauDesElementsSuivants
236
237             if (dizaine != '0') {
238                 FichierTexte_ecrireCaractere(fichier, dizaine);
239             }
240             FichierTexte_ecrireCaractere(fichier, unite);
241             Dictionnaire_sauvegarderElementsSuivants(fichier, tableauDesElementsSuivants[i]);
242         }
243     }
244
245
246     /* Partie publique */
247
248     /**
249     * \see TADDictionnaire.h
250     */
251     Dictionnaire Dictionnaire_dictionnaire() {
252         unsigned int i;
253         Dictionnaire dictionnaire;
254
255         for (i = 0; i < NB_LETTRES; ++i) {
256             dictionnaire.racines[i] = NULL;
257         }
```

```
258
259     dictionnaire.longueur = 0;
260
261     return dictionnaire;
262 } /* JC */
263
264 /**
265 * \see TADDictionnaire.h
266 */
267 unsigned int Dictionnaire_estVide(Dictionnaire dictionnaire) {
268     return (dictionnaire.longueur == 0);
269 } /* Antoine */
270
271 /**
272 * \see TADDictionnaire.h
273 */
274 void Dictionnaire_insererMot(Dictionnaire* dictionnaire, Mot mot) {
275     unsigned int i, j, longueur, indice;
276     ElementDictionnaire element, nouvelElement;
277     ElementDictionnaire tableauDesElementsSuivants[NB_LETTRES];
278
279     for (indice = 0; indice < NB_LETTRES; ++indice) {
280         tableauDesElementsSuivants[indice] = (dictionnaire->racines)[indice];
281     }
282     longueur = dictionnaire->longueur;
283
284     /* On insère le mot dans le dictionnaire */
285     for (i = 1; i <= Mot_longueur(mot); ++i) {
286         j = 0;
287         /* On vient chercher dans notre tableau le caractère correspondant au caractère de
288         while (j < longueur && (ElementDico_obtenirLettre(tableauDesElementsSuivants[j]) !=
289             j = j + 1;
290         }
291         /* Si on a trouvé, on va chercher dans le tableauDesElementsSuivants de l'élément,
292         if (j < longueur) {
293             element = tableauDesElementsSuivants[j];
294         }
295         /* Sinon on  ajoute l'élément */
296         else {
297             nouvelElement = ElementDico_element(Mot_iemeCaractere(mot, i), 0);
298
299             /* La racine n'existe pas dans le dictionnaire, on doit la créer */
300             if (i == 1) {
301                 dictionnaire->racines[dictionnaire->longueur] = nouvelElement;
302                 dictionnaire->longueur = dictionnaire->longueur + 1;
303             }
304             else {
305                 ElementDico_ajouterElementSuivant(&element, nouvelElement);
306             }
307             element = nouvelElement;
```

```
308         }
309
310         /* On passe à l'élément suivant pour le prochain caractère du mot */
311         for (indice = 0; indice < NB_LETTRES; ++indice) {
312             tableauDesElementsSuivants[indice] = (*ElementDico_obtenirTableauDesElementsSuiva
313         }
314         longueur = ElementDico_obtenirLongueurTableauDesElementsSuivants(element);
315     }
316
317     /* Enfin, on dit que le mot est présent dans le dictionnaire */
318     ElementDico_ajouterPresence(&element, 1);
319 }
320
321 /**
322 * \see TADDictionnaire.h
323 */
324 void Dictionnaire_insererFichier(Dictionnaire* dictionnaire, char* nomFichier) {
325     Fichier fichier;
326     char* ligne;
327
328     fichier = FichierTexte_fichierTexte(nomFichier);
329     FichierTexte_ouvrir(&fichier, LECTURE);
330     ligne = FichierTexte_lireChaine(fichier);
331
332     while (ligne != NULL)
333     {
334         Dictionnaire_insererMot(dictionnaire, Mot_genererMot(ligne));
335         free(ligne);
336         if (FichierTexte_finFichier(fichier) == 0) {
337             ligne = FichierTexte_lireChaine(fichier);
338         }
339         else
340             ligne = NULL;
341     }
342     free(ligne);
343     FichierTexte_fermer(&fichier);
344 } /* JC */
345
346 /**
347 * \see TADDictionnaire.h
348 */
349 unsigned int Dictionnaire_estPresent(Dictionnaire dictionnaire, Mot mot) {
350     unsigned int i, j, longueur, indice;
351     ElementDictionnaire tableauDesElementsSuivants[NB_LETTRES];
352     ElementDictionnaire element;
353
354     for (indice = 0; indice < NB_LETTRES; ++indice) {
355         tableauDesElementsSuivants[indice] = (dictionnaire.racines)[indice];
356     }
357     longueur = dictionnaire.longueur;
```

```
358
359     /* Vérifions que l'on trouve le mot caractère après caractère dans le dictionnaire */
360     for (i = 1; i <= Mot_longueur(mot); ++i) {
361         j = 0;
362         while (j < longueur && ElementDico_obtenirLettre(tableauDesElementsSuivants[j]) != M
363             j = j + 1;
364         }
365         /* On vérfie si on est sorti du while parce qu'on est arrivé à la fin du tableau ou
366         if (j < longueur && ElementDico_obtenirLettre(tableauDesElementsSuivants[j]) == Mot_
367             element = tableauDesElementsSuivants[j];
368         }
369         /* Echec, le mot recherché n'est pas dans le dictionnaire */
370         else {
371             return 0;
372         }
373
374         /* Tout va bien, on continue à avancer pour le prochain caractère */
375         for (indice = 0; indice < NB_LETTRES; ++indice) {
376             tableauDesElementsSuivants[indice] = (*ElementDico_obtenirTableauDesElementsSuiva
377         }
378         longueur = ElementDico_obtenirLongueurTableauDesElementsSuivants(element);
379     }
380
381     /* Si on n'a eu toujours aucune erreur, on retourne la valeur du booléen du dernier ca
382     return ElementDico_obtenirBooleen(element);
383 } /* Antoine */
384
385 /**
386 * \see TADDictionnaire.h
387 */
388 Dictionnaire Dictionnaire_charger(char* emplacement) {
389     Fichier fichier;
390     Dictionnaire dictionnaire;
391     unsigned int longueur, i;
392
393
394     dictionnaire = Dictionnaire_dictionnaire();
395     fichier = FichierTexte_fichierTexte(emplacement);
396     FichierTexte_ouvrir(&fichier, LECTURE);
397     longueur = Dictionnaire_recupererUneLongueur(fichier);
398     dictionnaire.longueur = longueur;
399
400     for (i = 0; i < longueur; ++i) {
401         dictionnaire.racines[i] = Dictionnaire_chargerElementsSuivants(fichier);
402     }
403     FichierTexte_fermer(&fichier);
404
405     return dictionnaire;
406 } /* Antoine */
407
```

```c
/**
* \see TADDictionnaire.h
*/
Fichier Dictionnaire_sauvegarder(Dictionnaire dictionnaire, char* emplacement) {
    Fichier fichier;
    char dizaine, unite;
    unsigned int i;

    fichier = FichierTexte_fichierTexte(emplacement);
    FichierTexte_ouvrir(&fichier, ECRITURE);
    Dictionnaire_naturelEnCaractere(dictionnaire.longueur, &dizaine, &unite);
    if (dizaine != '0') {
        FichierTexte_ecrireCaractere(&fichier, dizaine);
    }
    FichierTexte_ecrireCaractere(&fichier, unite);

    for (i = 0; i < dictionnaire.longueur; ++i)
    {
        FichierTexte_ecrireCaractere(&fichier, ElementDico_obtenirLettre(dictionnaire.racines
        FichierTexte_ecrireCaractere(&fichier, Dictionnaire_booleenEnCaractere(ElementDico_ob
        Dictionnaire_naturelEnCaractere(ElementDico_obtenirLongueurTableauDesElementsSuivants

        if (dizaine != '0') {
            FichierTexte_ecrireCaractere(&fichier, dizaine);
        }

        FichierTexte_ecrireCaractere(&fichier, unite);
        Dictionnaire_sauvegarderElementsSuivants(&fichier, dictionnaire.racines[i]);
    }
    FichierTexte_fermer(&fichier);

    return(fichier);
} /* JC */
```

## 3.3 FichierTexte

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <assert.h>
#include "TADFichierTexte.h"
#define NB_LETTRES_MAX 100

Fichier FichierTexte_fichierTexte(char* nomDuFichier) {
    Fichier fichier;
    fichier.file = NULL;
    fichier.nom = nomDuFichier;

    return fichier;
```

```
14  }
15
16  void FichierTexte_ouvrir(Fichier *fichier, Mode mode) {
17      assert(!FichierTexte_estOuvert(*fichier));
18
19      if (mode == ECRITURE) {
20              fichier->file = fopen(fichier->nom, "w+");
21              fichier->mode = mode;
22      }
23      else if (mode == LECTURE) {
24              fichier->file = fopen(fichier->nom, "r");
25              fichier->mode = mode;
26      }
27      else printf("pas le bon mode\n");
28
29  }
30
31  void FichierTexte_fermer(Fichier *fichier) {
32      if (FichierTexte_estOuvert(*fichier)) {
33          fclose(fichier->file);
34          fichier->file = NULL;
35      }
36  }
37
38  unsigned int FichierTexte_estOuvert(Fichier fichier) {
39      return fichier.file != NULL;
40  }
41
42  Mode FichierTexte_mode(Fichier fichier) {
43      return fichier.mode;
44  }
45
46  unsigned int FichierTexte_finFichier(Fichier fichier) {
47      /* Déclarations */
48      unsigned int resultat;
49      /* Préconditions */
50      assert((fichier.mode == LECTURE) && FichierTexte_estOuvert(fichier));
51
52      resultat = (fgetc(fichier.file)==EOF);
53      fseek(fichier.file, -1, SEEK_CUR);
54  /*      FichierTexte_deplacementCurseurMoinsUn(&fichier);
55  */
56      return (resultat);
57  }
58
59  void FichierTexte_ecrireChaine(Fichier *fichier, char* chaine) {
60      assert(FichierTexte_estOuvert(*fichier) && (FichierTexte_mode(*fichier) == ECRITURE));
61      fputs(chaine, fichier->file);
62  }
63
```

```c
char* FichierTexte_lireChaine(Fichier fichier) {
    assert(FichierTexte_estOuvert(fichier) && (FichierTexte_mode(fichier) == LECTURE) && !Fi

    char* chaine = malloc(NB_LETTRES_MAX * sizeof(char) +1 );
    fgets(chaine, NB_LETTRES_MAX, fichier.file);
    return chaine;
}

void FichierTexte_ecrireCaractere(Fichier *fichier, char caractere) {
    assert(FichierTexte_estOuvert(*fichier) && (FichierTexte_mode(*fichier) == ECRITURE));
    fputc(caractere, fichier->file);
}

char FichierTexte_lireCaractere(Fichier fichier) {
    assert(FichierTexte_estOuvert(fichier) && (FichierTexte_mode(fichier) == LECTURE) && !Fi
    return fgetc(fichier.file);
}

void FichierTexte_deplacementCurseurMoinsUn(Fichier *fichier) {
    assert(FichierTexte_estOuvert(*fichier) && (FichierTexte_mode(*fichier) == LECTURE));
    fseek(fichier->file,-1, SEEK_CUR );
}

unsigned int FichierTexte_existe(char* emplacement) {
    if (access(emplacement, F_OK ) != -1)
        return 1;
    else
        return 0;
}

unsigned int FichierTexte_nombreDeCaractere(Fichier fichier){
    /* Déclarations */
    unsigned int resultat;
    char caractereTemp;

    /* Corps */
    resultat = 0;
    while (!FichierTexte_finFichier(fichier)) {
        caractereTemp = FichierTexte_lireCaractere(fichier);
        if (!FichierTexte_estUnRetourChariot(caractereTemp))
            resultat = resultat + 1;
    }
    return resultat;
}

unsigned int FichierTexte_estUnRetourChariot(char caractere){
    return (caractere == '\n');
}
```

## 3.4 ListeDeMot

```c
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <string.h>
#include "TADMot.h"
#include "TADListeDeMotPrive.h"
#include "TADListeDeMot.h"

/* On respecte la conception du TAD Liste : un tableau et une longueur */

/*
*    PUBLIC
*/
ListeDeMot ListeDeMot_liste() {
    ListeDeMot l;
    l.tableau = NULL;
    l.tableau = malloc(sizeof(Mot));
    l.longueur = 0;

    return l;
}

int ListeDeMot_estVide(ListeDeMot liste) {
    return (liste.longueur == 0);
}

unsigned int ListeDeMot_longueur(ListeDeMot liste) {
    return liste.longueur;
}

void ListeDeMot_ajouter(ListeDeMot *liste, Mot mot) {
    liste->longueur = liste->longueur + 1;
    liste->tableau = realloc(liste->tableau, sizeof(Mot) * liste->longueur);
    liste->tableau[liste->longueur - 1] = mot;
}

Mot ListeDeMot_obtenir(ListeDeMot liste, unsigned int indice) {
    assert(1 <= indice && indice <= ListeDeMot_longueur(liste));

    return liste.tableau[indice - 1];
}

void ListeDeMot_supprimer(ListeDeMot* liste, unsigned int indice) {
    assert(1 <= indice && indice <= ListeDeMot_longueur(*liste));

    ListeDeMot_decalerAGauche(liste, indice);
    liste->longueur = liste->longueur - 1;
    liste->tableau = realloc(liste->tableau, sizeof(Mot) * liste->longueur);
```

```c
}

void ListeDeMot_inserer(ListeDeMot *liste, Mot mot, unsigned int indice) {
    assert(1 <= indice && indice <= ListeDeMot_longueur(*liste) + 1);

    liste->longueur = liste->longueur + 1;
    liste->tableau = realloc(liste->tableau, sizeof(Mot) * liste->longueur);
    ListeDeMot_decalerADroite(liste, indice);
    liste->tableau[indice - 1] = mot;
}

void ListeDeMot_listeDeMotEnChaine(ListeDeMot liste) {
    int i;

    for(i = 1; i <= ListeDeMot_longueur(liste); i++){
        printf("%s ", Mot_obtenirContenu(ListeDeMot_obtenir(liste, i)));
    }
}

void ListeDeMot_liberer(ListeDeMot *liste){
    free(liste->tableau);
}

unsigned int ListeDeMot_estPresent(ListeDeMot liste, Mot mot){
    unsigned int i, res;

    i = 1;
    res = 0;

    while (!res && i <= ListeDeMot_longueur(liste)) {
        res = Mot_sontEgaux(ListeDeMot_obtenir(liste, i), mot);
        i++;
    }

    return res;
}


/*
*    PRIVÉE
*/
void ListeDeMot_decalerADroite(ListeDeMot *liste, unsigned int indice) {
    /* Déclarations */
    unsigned int i;

    /* Corps */
    for (i = liste->longueur - 1; i >= indice; i--) {
        liste->tableau[i] = liste->tableau[i-1];
    }
}
```

```
99
100 void ListeDeMot_decalerAGauche(ListeDeMot *liste, unsigned int indice) {
101     /* Déclarations */
102     unsigned int i;
103
104     /* Déclarations */
105     for (i = indice - 1; i < liste->longueur; i++) {
106         liste->tableau[i] = liste->tableau[i + 1];
107     }
108 }
```

## 3.5 ListeDeSuperMot

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4  #include "TADSuperMot.h"
5  #include "TADListeDeSuperMot.h"
6  #include "TADListeDeSuperMotPrive.h"
7
8  /* On respecte la conception du TAD Liste : un tableau et une longueur */
9
10 /*
11 *    PUBLIC
12 */
13 ListeDeSuperMot ListeDeSuperMot_liste() {
14     ListeDeSuperMot l;
15
16     l.tableau = NULL;
17     l.tableau = malloc(sizeof(SuperMot));
18     l.longueur = 0;
19
20     return l;
21 }
22
23 int ListeDeSuperMot_estVide(ListeDeSuperMot liste) {
24     return (liste.longueur == 0);
25 }
26
27 unsigned int ListeDeSuperMot_longueur(ListeDeSuperMot liste) {
28     return liste.longueur;
29 }
30
31 void ListeDeSuperMot_ajouter(ListeDeSuperMot *liste, SuperMot superMot) {
32
33     liste->longueur = liste->longueur + 1;
34     liste->tableau = realloc(liste->tableau, sizeof(SuperMot) * liste->longueur);
35     liste->tableau[liste->longueur - 1] = superMot;
36 }
```

```
37
38 SuperMot ListeDeSuperMot_obtenir(ListeDeSuperMot liste, unsigned int indice) {
39     assert(1 <= indice && indice <= ListeDeSuperMot_longueur(liste));
40
41     return liste.tableau[indice - 1];
42 }
43
44 void ListeDeSuperMot_supprimer(ListeDeSuperMot* liste, unsigned int indice) {
45     assert(1 <= indice && indice <= ListeDeSuperMot_longueur(*liste));
46
47     ListeDeSuperMot_decalerAGauche(liste, indice);
48     liste->longueur = liste->longueur - 1;
49     liste->tableau = realloc(liste->tableau, sizeof(SuperMot) * liste->longueur);
50 }
51
52 void ListeDeSuperMot_inserer(ListeDeSuperMot *liste, SuperMot superMot, unsigned int indice)
53     assert(1 <= indice && indice <= ListeDeSuperMot_longueur(*liste) + 1);
54
55     liste->longueur = liste->longueur + 1;
56     liste->tableau = realloc(liste->tableau, sizeof(SuperMot) * liste->longueur);
57     ListeDeSuperMot_decalerADroite(liste, indice);
58     liste->tableau[indice - 1] = superMot;
59 }
60
61 void ListeDeSuperMot_liberer(ListeDeSuperMot *liste){
62     free(liste->tableau);
63 }
64 /*
65 *    PRIVÉE
66 */
67 void ListeDeSuperMot_decalerADroite(ListeDeSuperMot *liste, unsigned int indice) {
68     unsigned int i;
69
70     for (i = liste->longueur - 1; i >= indice; i--) {
71         liste->tableau[i] = liste->tableau[i - 1];
72     }
73 }
74
75 void ListeDeSuperMot_decalerAGauche(ListeDeSuperMot *liste, unsigned int indice) {
76     unsigned int i;
77
78     for (i = indice - 1; i < liste->longueur; i++) {
79             liste->tableau[i] = liste->tableau[i + 1];
80     }
81 }
```

## 3.6 Mot

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#include "TADMot.h"
#include "TADMotPrive.h"

/*
*    PUBLIC
*/
Mot Mot_mot() {

    Mot mot;
    mot.contenu = malloc(sizeof(char));
    mot.contenu[0] = '\0';
    mot.longueur = 0;

    return mot;
}

char* Mot_obtenirContenu(Mot mot) {
    return mot.contenu;
}

unsigned int Mot_longueur(Mot mot) {
    return mot.longueur;
}

Mot Mot_genererMot(char* chaine) {

    Mot mot = Mot_mot();
    for (int i = 1; i <= strlen(chaine); i++) {

        if (estLettreOuTiret(chaine[i - 1])) {
            Mot_insererCaractere(&mot, chaine[i - 1], i);
        }

    }

    return mot;
}

unsigned int Mot_sontEgaux(Mot mot1, Mot mot2) {

    if (mot1.longueur != mot2.longueur)
        return 0;
    else {
```

```
49          if (strcmp(mot1.contenu, mot2.contenu) == 0)
50              return 1;
51          else
52              return 0;
53      }
54  }
55
56  void Mot_insererCaractere(Mot* mot, char c, unsigned int position) {
57      assert(0 < position && position <= (Mot_longueur(*mot) + 1) && estLettreOuTiret(c));
58
59      if (position == (Mot_longueur(*mot) + 1)) {
60
61          mot->longueur = mot->longueur + 1;
62          mot->contenu = realloc(mot->contenu, sizeof(char) * (mot->longueur + 1));
63          mot->contenu[position - 1] = c;
64          mot->contenu[position] = '\0';
65      }
66      else {
67
68          for (int i = Mot_longueur(*mot) + 1; i >= (position + 1); i--) {
69              Mot_modifierCaractere(mot, Mot_iemeCaractere(*mot, i - 1), i);
70          }
71          mot->longueur = mot->longueur + 1;
72          Mot_modifierCaractere(mot, c, position);
73      }
74  }
75
76  void Mot_modifierCaractere(Mot* mot, char c, unsigned int position) {
77      assert(0 < position && position <= (Mot_longueur(*mot) + 1) && estLettreOuTiret(c));
78
79      mot->contenu[position - 1] = c;
80
81      if (position == Mot_longueur(*mot) + 1)
82          mot->contenu[position] = '\0';
83  }
84
85  void Mot_supprimerCaractere(Mot* mot, unsigned int position) {
86      assert(0 < position && position <= Mot_longueur(*mot));
87
88      char c;
89
90      if (mot->longueur == 1 && position == 1) {
91          *mot = Mot_mot();
92      }
93      else {
94          for (int i = position; i < Mot_longueur(*mot); ++i) {
95              c = Mot_iemeCaractere(*mot, i + 1);
96              Mot_modifierCaractere(mot, c, i);
97          }
98          mot->longueur = mot->longueur - 1;
```

```
99      }
100     mot->contenu[Mot_longueur(*mot)] = '\0';
101     mot->contenu = realloc(mot->contenu, sizeof(char) * (mot->longueur + 1));
102 }
103
104 char Mot_iemeCaractere(Mot mot, unsigned int position) {
105     assert(0 < position && position < Mot_longueur(mot) + 1);
106
107     return mot.contenu[position - 1];
108 }
109
110 Mot Mot_concatener(Mot mot1, Mot mot2) {
111
112     return Mot_genererMot(strcat(mot1.contenu, mot2.contenu));
113 }
114
115 unsigned int Mot_estUnMotValide(Mot mot) {
116
117     int i = 1;
118     int res = 1;
119
120     while (res && i <= Mot_longueur(mot)) {
121         res = estLettreOuTiret(Mot_iemeCaractere(mot, i));
122         i = i + 1;
123     }
124
125     return res;
126 }
127
128 void Mot_separerMots(Mot mot, unsigned int position, Mot* mot1, Mot* mot2) {
129     assert(2 <= position && position <= Mot_longueur(mot));
130
131     char c;
132
133     *mot1 = Mot_mot();
134
135     for (int i = 1; i < position; ++i) {
136         c = Mot_iemeCaractere(mot, i);
137         Mot_insererCaractere(mot1, c, i);
138     }
139
140     *mot2 = Mot_mot();
141     for (int j = position; j <= Mot_longueur(mot); ++j) {
142         c = Mot_iemeCaractere(mot, j);
143         Mot_insererCaractere(mot2, c, j - position + 1);
144     }
145 }
146
147 Mot Mot_sousMot(Mot mot, unsigned int debut, unsigned int longueur) {
148     assert(0 < debut && 0 < longueur && (debut + longueur - 1) <= Mot_longueur(mot) && Mot_l
```

```
149
150     char c;
151     Mot nouveau;
152
153     nouveau = Mot_mot();
154     for (int i = debut; i < debut + longueur; ++i) {
155         c = Mot_iemeCaractere(mot, i);
156         Mot_insererCaractere(&nouveau, c, i - debut + 1);
157     }
158
159     return nouveau;
160 }
161
162 unsigned int Mot_estSousMot(Mot mot1, Mot mot2) {
163
164     if (Mot_longueur(mot1) > Mot_longueur(mot2))
165         return 0;
166     else {
167         if (strstr(mot2.contenu, mot1.contenu) == NULL)
168             return 0;
169         else
170             return 1;
171     }
172 }
173
174 unsigned int Mot_sontIdentiques(Mot* mot1, Mot* mot2) {
175
176     return (mot1 == mot2);
177 }
178
179 /*
180 *    PRIVÉE
181 */
182 unsigned int max(unsigned int a, unsigned int b) {
183
184     if (a > b)
185         return a;
186     else
187         return b;
188 }
189
190 unsigned int estLettreOuTiret(char c) {
191     return (isalpha(c) || c == '-' || c == 'à' || c == 'é' || c == 'è' || c == 'ç' || c == 
192 }
```

## 3.7 Parametres

```
1 /* Ce commentaire est utilisé pour forcer la conversion en ISO */
2 #include <stdio.h>
```

```c
#include <stdlib.h>
#include <assert.h>
#include "TADParametres.h"

Parametres Parametres_parametre(){
    Parametres p;

    p.dictionnaire = NULL;
    p.fichier = NULL;
    p.cible = NULL;
    p.aide = 0;
    return p;
}

void Parametres_fixerDictionnaire(Parametres* p, char* nomDuDico){
    (*p).dictionnaire = nomDuDico;
}

void Parametres_fixerFichier(Parametres* p, char* nomDuFichier){
    (*p).fichier = nomDuFichier;
}

void Parametres_fixerCible(Parametres* p, char* nomDeLaCible){
    (*p).cible = nomDeLaCible;
}

void Parametres_fixerAide(Parametres* p, int presenceDuH){
    (*p).aide = presenceDuH;
}

char* Parametres_obtenirDictionnaire(Parametres p){
    return p.dictionnaire;
}

char* Parametres_obtenirFichier(Parametres p){
    return p.fichier;
}

int Parametres_obtenirAide(Parametres p){
    return p.aide;
}

char* Parametres_obtenirCible(Parametres p){
    return p.cible;
}
```

## 3.8 SuperMot

```c
/* Ce commentaire est utilisé pour forcer la conversion en ISO */
#include <stdio.h>
#include <stdlib.h>
#include "TADSuperMot.h"
#include "TADMot.h"
#include "TADListeDeMot.h"

SuperMot SuperMot_superMot() {
    SuperMot superMot;

    superMot.mot = Mot_mot();
    superMot.position = 1;
    superMot.listeDeCorrections = ListeDeMot_liste();
    superMot.valide = 0;

    return superMot;
}

Mot SuperMot_obtenirMot(SuperMot superMot) {
    return superMot.mot;
}

void SuperMot_fixerMot(SuperMot* superMot, Mot mot) {
    (*superMot).mot = mot;
}

unsigned int SuperMot_obtenirPosition(SuperMot superMot) {
    return superMot.position;
}

void SuperMot_fixerPosition(SuperMot* superMot, unsigned int position) {
    (*superMot).position = position;
}

ListeDeMot SuperMot_obtenirListeDeCorrection(SuperMot superMot) {
    return superMot.listeDeCorrections;
}

void SuperMot_fixerListeDeCorrections(SuperMot* superMot, ListeDeMot corrections) {
    (*superMot).listeDeCorrections = corrections;
}

int SuperMot_obtenirValidite(SuperMot superMot) {
    return superMot.valide;
}

void SuperMot_fixerValidite(SuperMot* superMot, int estValide) {
    (*superMot).valide = estValide;
```

```
49 }
```

# Chapitre 4

# Fichiers sources de la logique métier

## Sommaire



## 4.1 affichage

```c
#include <stdio.h>
#include <stdlib.h>
#include "affichage.h"


void IHM_afficherAide() {
    printf("Aide Asispell :\n");
    printf("    asispell -h : cette aide\n");
    printf("    asispell -d dico : correction de l'entrée standard à l'aide du dico\n");
    printf("    asispell -h dico -f fic : compléter le dictionnaire dico à l'aide des mots du
}

void IHM_afficherResultat(int erreur, char* resultat) {
    if (erreur != 0)
        IHM_afficherErreur(erreur);
    else
        IHM_afficherCorrection(resultat);
}

void IHM_afficherCorrection(char* resultat) {
    printf("%s\n", resultat);
}

void IHM_afficherErreur(int erreur) {

    switch(erreur) {
        case 1 :
```

```
28              printf("Au moins un caractère entré n'est pas valide.");
29              break;
30
31          case 2 :
32              printf("erreur de type 2");
33              break;
34
35          default :
36              printf("erreur par défaut");
37      }
38  }
```

## 4.2 corriger

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <string.h>
4   #include <ctype.h>
5   #include "corriger.h"
6   #include "TADMot.h"
7   #include "TADSuperMot.h"
8   #include "TADDictionnaire.h"
9   #include "TADCorrecteurOrthographique.h"
10  #include "TADListeDeSuperMot.h"
11  #include "math.h"
12  #include "transtypage.h"
13  #include "LMSuperMot.h"
14  #include "affichage.h"
15
16  /**
17  * PRIVE
18  */
19  int estCaractereValide(char c) {
20      return (isalpha(c) || c == '-' || c == 'à' || c == 'é' || c == 'è' || c == 'ç' || c == 
21  }
22
23  /**
24  * \fn void erreurAllocationMemoire()
25  * \brief Affiche une erreur dans l'entrée standard lors d'une allocation mémoire qui n'a p
26  * \warning Fonction privée
27  */
28  void erreurAllocationMemoire() {
29      fprintf(stderr, "Erreur d'allocation mémoire\n");
30      exit(1);
31  }
32
33  /* Corriger l'analyse descendante*/
34  void corriger(char* chaine, Dictionnaire dictionnaire) {
35      unsigned int estValide, erreur;
```

```
36      ListeDeSuperMot listeDeSuperMots;
37      estValide = 0;
38      erreur = 0;
39
40      evaluerSaisie(&estValide, &erreur, chaine);
41
42      if (estValide) {
43          listeDeSuperMots = ListeDeSuperMot_liste();
44          listeDeSuperMots = genererListeDeSuperMots(chaine, dictionnaire);
45          corrigerListeDeSuperMots(&listeDeSuperMots, dictionnaire);
46          afficherCorrection(listeDeSuperMots);
47      }
48      else
49          IHM_afficherErreur(erreur);
50  }
51
52  /**
53  * \see corriger.h
54  */
55  void evaluerSaisie(unsigned int* estValide,unsigned int* erreur, char* chaine) {
56      int i = 0;
57      *estValide = 1;
58
59      while (i < strlen(chaine) && *estValide == 1) {
60          if (!estCaractereValide(chaine[i])) {
61              *estValide = 0;
62              *erreur = 1;
63          }
64          i++;
65      }
66  }
67
68  /**
69  * \see corriger.h
70  */
71  char* lireChaine() {
72      int max = 20;
73      char c;
74      // On alloue la mémoire
75      char* name = (char*) malloc(max);
76      int i = 0;
77
78      if (name == 0)
79          erreurAllocationMemoire();
80
81      // On saute les premiers espaces
82      while (1) {
83          c = getchar();
84          if (c == EOF)
85              break;
```

```
86          if (!isspace(c)) {
87              ungetc(c, stdin);
88              break;
89          }
90      }
91
92      while (1) {
93          c = getchar();
94          // À la fin de la chaîne on rajoute le caractère de fin
95          if (c == '.' || c == EOF) {
96              name[i] = '\0';
97              break;
98          }
99          name[i] = c;
100         // Le buffer est plein
101         if (i == max - 1) {
102             max = max + max;
103             // On crée un buffer plus grand
104             name = (char*) realloc(name, max);
105             if (name == 0)
106                 erreurAllocationMemoire();
107         }
108         i++;
109     }
110
111     return name;
112 }
113
114 /**
115 * \see corriger.h
116 */
117 void corrigerListeDeSuperMots(ListeDeSuperMot* listeDeSuperMots, Dictionnaire dictionnaire)
118     int i;
119     SuperMot superMotTemp;
120
121     for (i = 1; i <= ListeDeSuperMot_longueur(*listeDeSuperMots); i++) {
122         superMotTemp = ListeDeSuperMot_obtenir(*listeDeSuperMots, i);
123         if (!SuperMot_obtenirValidite(superMotTemp)) {
124             SuperMot_fixerListeDeCorrections(&superMotTemp, CorrecteurOrthographique_propose
125             ListeDeSuperMot_supprimer(listeDeSuperMots, i);
126             ListeDeSuperMot_inserer(listeDeSuperMots, superMotTemp, i);
127         }
128     }
129 }
130
131 void afficherCorrection(ListeDeSuperMot listeDeSuperMots) {
132     int i;
133     SuperMot superMotTemp;
134
135     for (i = 1; i <= ListeDeSuperMot_longueur(listeDeSuperMots); i++) {
```

```
136         superMotTemp = ListeDeSuperMot_obtenir(listeDeSuperMots, i);
137 
138         /* Si le superMot courant est valide */
139         if (SuperMot_obtenirValidite(superMotTemp)) {
140             printf("*");
141         }
142 
143         else {
144             printf("& %s %d %d : ",
145                 Mot_obtenirContenu(SuperMot_obtenirMot(superMotTemp)),
146                 ListeDeMot_longueur(SuperMot_obtenirListeDeCorrection(superMotTemp)),
147                 SuperMot_obtenirPosition(superMotTemp)
148             );
149             ListeDeMot_listeDeMotEnChaine(SuperMot_obtenirListeDeCorrection(superMotTemp));
150         }
151         printf("\n");
152     }
153 }
```

## 4.3 main

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include "TADParametres.h"
5  #include "TADDictionnaire.h"
6  #include "affichage.h"
7  #include "corriger.h"
8 
9  /**
10 * \fn Parametre formaterParametres(unsigned int nbParametres, char* parametres) {
11 * \brief Renvoie une structure paramètre permettant de savoir facilement quelles sont les
12 * \param unsigned int nbParametres : le nombre de paramètres
13 * \param char* parametres : la liste de paramètres passés
14 * \return Parametre parametre
15 * \author Étienne Batise
16 */
17 Parametres formaterParametres(unsigned int nbParametres, char* const* parametres) {
18     int optch;
19     char format[] = "hd:f:";
20     Parametres parametre;
21 
22     parametre = Parametres_parametre();
23     while ((optch = getopt(nbParametres, parametres, format)) != -1) {
24         switch (optch) {
25             case 'h':
26                 Parametres_fixerAide(&parametre, 1);
27             break;
28 
```

```
29              case 'd':
30                  Parametres_fixerDictionnaire(&parametre, optarg);
31              break;
32
33              case 'f':
34                  Parametres_fixerFichier(&parametre, optarg);
35              break;
36          }
37      }
38
39      return parametre;
40  }
41
42  /**
43  * \fn void executerProgramme(Parametre parametre)
44  * \brief Execute le programme en fonction des champs de parametre remplies
45  * \param Parametre parametre : la structure dans les champs permettent de lancer tell ou t
46  * \author Étienne Batise
47  */
48  void executerProgramme(Parametres parametre) {
49      Dictionnaire d;
50      char* entreeStandard;
51
52      d = Dictionnaire_dictionnaire();
53
54      if ((Parametres_obtenirAide(parametre) == 1)) { //|| ((Parametres_obtenirFichier(parame
55          IHM_afficherAide();
56      }
57      else {
58
59          /* Si on a un fichier et un dictionnaire, on complète le dictionnaire déjà sérialis
60          if ((Parametres_obtenirFichier(parametre) != NULL) && (Parametres_obtenirDictionnair
61              if (FichierTexte_existe(Parametres_obtenirDictionnaire(parametre))) {
62                  if (FichierTexte_existe(Parametres_obtenirFichier(parametre))) {
63                      d = Dictionnaire_charger(Parametres_obtenirDictionnaire(parametre));
64                      Dictionnaire_insererFichier(&d, Parametres_obtenirFichier(parametre));
65                      Dictionnaire_sauvegarder(d, Parametres_obtenirDictionnaire(parametre));
66
67                      printf("Le fichier %s a bien été ajouté au dictionnaire %s.\n", Parametr
68                  }
69                  else
70                      printf("Erreur : le fichier %s demandé n'existe pas.\n", Parametres_obte
71              }
72              else
73                  printf("Erreur : le dictionnaire %s demandé n'existe pas.\n", Parametres_obt
74          }
75          else {
76              entreeStandard = lireChaine();
77              Parametres_fixerCible(&parametre, entreeStandard);
78
```

```
79              /* On effectue la correction avec le dictionnaire déjà sérialisé */
80              if ((Parametres_obtenirCible(parametre) != NULL) && (Parametres_obtenirDictionna
81                  if (FichierTexte_existe(Parametres_obtenirDictionnaire(parametre))) {
82                      d = Dictionnaire_charger(Parametres_obtenirDictionnaire(parametre));
83                      corriger(Parametres_obtenirCible(parametre), d);
84                  }
85                  else
86                      printf("Erreur : le dictionnaire %s demandé n'existe pas.\n", Parametres
87
88              }
89              free(entreeStandard);
90          }
91
92      }
93  }
94
95  int main(int argc, char *argv[]) {
96      Parametres parametres;
97
98      parametres = formaterParametres(argc, argv);
99      executerProgramme(parametres);
100
101     return 0;
102 }
```

## 4.4 transtypage

```
1  /* Ce commentaire est utilisé pour forcer la conversion en ISO */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #include "transtypage.h"
6
7  char* Transtypage_entierEnChaine(unsigned int e){
8      int nouveauE, indice, taille, longueur, temp, p;
9      char* retour;
10     nouveauE = e;
11     taille = 1;
12     longueur = 1;
13     indice  = 0;
14     while (nouveauE / longueur >= 10) {
15         longueur = longueur * 10;
16         taille = taille + 1;
17     }
18
19     retour = malloc(sizeof(char) * (taille + 1));
20
21     for (p = taille; p >= 1; p--) {
22         temp = nouveauE / pow(10, p - 1);
```

```
23          retour[indice] = (char)(((int)'0') + temp);
24          indice = indice + 1;
25          nouveauE = nouveauE - temp * pow(10, p - 1);
26      }
27
28      return retour;
29  }
```

# Chapitre 5

# Fichiers sources des tests unitaires

## Sommaire



## 5.1 Dictionnaire

```c
#include <stdio.h>
#include <stdlib.h>
#include <CUnit/Basic.h>
#include <string.h>
#include "TADFichierTexte.h"
#include "TADDictionnaire.h"
#include "TADElementDictionnairePrive.h"
#include "TADMot.h"

/*
Compilation : make TUdico
*/

int init_suite_success (void) {
    return 0;
}

int clean_suite_success (void) {
    return 0;
}

void test_dictionnaire(void) {
    CU_ASSERT_TRUE(0);
```

```
}

void test_creationElement(void) {
    ElementDictionnaire element = ElementDico_element('c', 1);

    CU_ASSERT_EQUAL(ElementDico_obtenirLettre(element), 'c');
    CU_ASSERT_TRUE(ElementDico_obtenirBooleen(element));
    CU_ASSERT_EQUAL(ElementDico_obtenirLongueurTableauDesElementsSuivants(element), 0);

    ElementDico_ajouterPresence(&element, 0);
    CU_ASSERT_FALSE(ElementDico_obtenirBooleen(element));

    ElementDico_ajouterElementSuivant(&element, ElementDico_element('a', 1));
    CU_ASSERT_EQUAL(ElementDico_obtenirLongueurTableauDesElementsSuivants(element), 1);

    ElementDico_supprimerElementSuivant(&element, 0);
    CU_ASSERT_EQUAL(ElementDico_obtenirLongueurTableauDesElementsSuivants(element), 0);
}

void test_estVide(void) {
    /* On crée un dictionnaire d et on regarde si il est vide */
    Dictionnaire d = Dictionnaire_dictionnaire();
    CU_ASSERT_TRUE(Dictionnaire_estVide(d));

    /* On crée un mot qu'on insère dans le dictionnaire d et on regarde si il n'est pas vi
    Mot m = Mot_genererMot("Salut");
    Dictionnaire_insererMot(&d, m);
    CU_ASSERT_FALSE(Dictionnaire_estVide(d));
}

void test_insererMot(void) {
    Mot m = Mot_genererMot("Salut");
    Mot m2 = Mot_genererMot("Bonjour");
    Mot m3 = Mot_genererMot("Salutations");
    Mot m4 = Mot_genererMot("Zebre");
    Mot m5 = Mot_genererMot("Pique-nique");

    for (int i = 1; i <= 4; ++i)
    {
        Dictionnaire d =  Dictionnaire_dictionnaire();
        CU_ASSERT_TRUE(Dictionnaire_estVide(d));

        if (i == 1) {
            Dictionnaire_insererMot(&d, m);
            Dictionnaire_insererMot(&d, m2);
            Dictionnaire_insererMot(&d, m3);
            Dictionnaire_insererMot(&d, m4);
            Dictionnaire_insererMot(&d, m5);
        }
        else if (i == 2) {
```

```
74              Dictionnaire_insererMot(&d, m3);
75              Dictionnaire_insererMot(&d, m);
76              Dictionnaire_insererMot(&d, m4);
77              Dictionnaire_insererMot(&d, m5);
78              Dictionnaire_insererMot(&d, m2);
79          }
80          else if (i == 3) {
81              Dictionnaire_insererMot(&d, m4);
82              Dictionnaire_insererMot(&d, m);
83              Dictionnaire_insererMot(&d, m3);
84              Dictionnaire_insererMot(&d, m2);
85              Dictionnaire_insererMot(&d, m5);
86          }
87          else if (i == 4) {
88              Dictionnaire_insererMot(&d, m5);
89              Dictionnaire_insererMot(&d, m4);
90              Dictionnaire_insererMot(&d, m3);
91              Dictionnaire_insererMot(&d, m2);
92              Dictionnaire_insererMot(&d, m);
93          }
94
95          CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m));
96          CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m2));
97          CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m3));
98          CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m4));
99          CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m5));
100         CU_ASSERT_FALSE(Dictionnaire_estVide(d));
101     }
102 }
103
104 void test_insererFichier(void) {
105     /* On crée un fichierTexte qu'on ouvre en écriture */
106     Fichier f = FichierTexte_fichierTexte("tests/testTADDictionnaire.txt");
107     FichierTexte_ouvrir(&f, ECRITURE);
108     /* On écrit 'a' dans le fichier et on ferme le fichier */
109     FichierTexte_ecrireCaractere(&f, 'a');
110     FichierTexte_fermer(&f);
111
112     /* On crée un dictionnaire et deux Mots */
113     Dictionnaire d = Dictionnaire_dictionnaire();
114     Mot m = Mot_genererMot("a");
115     Mot m1 = Mot_genererMot("b");
116
117     /* On insère le fichier créé avant dans le dictionnaire */
118     Dictionnaire_insererFichier(&d, "tests/testTADDictionnaire.txt");
119
120     /* On vérifie qu'il n'est pas vide, qu'il contient le mot correspondant au caractère e
121     CU_ASSERT_FALSE(Dictionnaire_estVide(d));
122     CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m));
123     CU_ASSERT_FALSE(Dictionnaire_estPresent(d, m1));
```

```
124
125     /* On insère un fichier comportant 3 mots, on vérifie que le dictionnaire contient les
126     Dictionnaire d3 = Dictionnaire_dictionnaire();
127     Mot m2 = Mot_genererMot("Bonjour");
128     Mot m3 = Mot_genererMot("Monsieur");
129     Mot m4 = Mot_genererMot("Aurevoir");
130
131     Dictionnaire_insererFichier(&d3, "tests/testTADDictionnaire5.txt");
132
133     CU_ASSERT_TRUE(Dictionnaire_estPresent(d3, m2));
134     CU_ASSERT_TRUE(Dictionnaire_estPresent(d3, m3));
135     CU_ASSERT_TRUE(Dictionnaire_estPresent(d3, m4));
136     CU_ASSERT_FALSE(Dictionnaire_estVide(d3));
137
138     Dictionnaire d4 = Dictionnaire_dictionnaire();
139     Dictionnaire_insererFichier(&d4, "tests/testTADDictionnaire7.txt");
140     CU_ASSERT_TRUE(d4.longueur == 10);
141
142     /* On sauvegarde le dictionnaire */
143     Dictionnaire_sauvegarder(d4, "tests/testTADDictionnaire8.txt");
144
145     /* On charge un dictionnaire sérialisé et on insère un fichier contenant des mots, on
146     des mots est contenu dans le dictionnaire */
147     Dictionnaire d5 = Dictionnaire_charger("tests/testTADDictionnaire8.txt");
148     Dictionnaire_insererFichier(&d5, "tests/testTADDictionnaire9.txt");
149
150     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("Bonjour")));
151     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("Monsieur")));
152     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("Comment")));
153     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("allez")));
154     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("vous")));
155     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("et")));
156     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("votre")));
157     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("Femme")));
158     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("merveilleusement")));
159     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("bien")));
160     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("malheureusement")));
161     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("Aurevoir")));
162     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("ça")));
163     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("va")));
164     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("été")));
165     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("sommes")));
166     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("bientôt")));
167     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("élève")));
168     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("noël")));
169     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("ççdçççéeéé")));
170
171     Dictionnaire dicoASI = Dictionnaire_dictionnaire();
172     Dictionnaire_insererFichier(&dicoASI, "tests/dico-ref-ascii.txt");
173     Dictionnaire_sauvegarder(dicoASI, "tests/dico.dico");
```

```
174 }
175
176 void test_charger(void) {
177     Fichier f = FichierTexte_fichierTexte("tests/testTADDictionnaire3.txt");
178     FichierTexte_ouvrir(&f, ECRITURE);
179     FichierTexte_ecrireCaractere(&f, '1');
180     FichierTexte_ecrireCaractere(&f, 'S');
181     FichierTexte_ecrireCaractere(&f, '0');
182     FichierTexte_ecrireCaractere(&f, '1');
183     FichierTexte_ecrireCaractere(&f, 'a');
184     FichierTexte_ecrireCaractere(&f, '0');
185     FichierTexte_ecrireCaractere(&f, '1');
186     FichierTexte_ecrireCaractere(&f, 'l');
187     FichierTexte_ecrireCaractere(&f, '1');
188     FichierTexte_ecrireCaractere(&f, '0');
189     FichierTexte_fermer(&f);
190
191     Dictionnaire d = Dictionnaire_charger("tests/testTADDictionnaire3.txt");
192     Mot mot = Mot_genererMot("Sal");
193
194     CU_ASSERT_TRUE(Dictionnaire_estPresent(d, mot));
195
196     Dictionnaire d2 = Dictionnaire_dictionnaire();
197     Mot m2 = Mot_genererMot("Bonjour");
198     Mot m3 = Mot_genererMot("Monsieur");
199     Mot m4 = Mot_genererMot("Aurevoir");
200
201     Dictionnaire_insererFichier(&d2, "tests/testTADDictionnaire5.txt");
202     Dictionnaire_sauvegarder(d2, "tests/testTADDictionnaire6.txt");
203
204     Dictionnaire d3 = Dictionnaire_charger("tests/testTADDictionnaire6.txt");
205
206     CU_ASSERT_TRUE(Dictionnaire_estPresent(d3, m2));
207     CU_ASSERT_TRUE(Dictionnaire_estPresent(d3, m3));
208     CU_ASSERT_TRUE(Dictionnaire_estPresent(d3, m4));
209
210     Dictionnaire d4 = Dictionnaire_charger("tests/testTADDictionnaire8.txt");
211     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("Bonjour")));
212     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("Monsieur")));
213     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("Comment")));
214     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("allez")));
215     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("vous")));
216     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("et")));
217     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("votre")));
218     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("Femme")));
219     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("merveilleusement")));
220     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("bien")));
221     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("malheureusement")));
222     CU_ASSERT_TRUE(Dictionnaire_estPresent(d4, Mot_genererMot("Aurevoir")));
223
```

```
224     Dictionnaire d5 = Dictionnaire_dictionnaire();
225     Dictionnaire_insererFichier(&d5, "tests/testTADDictionnaire9.txt");
226 
227     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("ça")));
228     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("va")));
229     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("été")));
230     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("sommes")));
231     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("bientôt")));
232     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("élève")));
233     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("noël")));
234     CU_ASSERT_TRUE(Dictionnaire_estPresent(d5, Mot_genererMot("ççdçççéeéé")));
235 
236     Dictionnaire_sauvegarder(d5, "tests/testTADDictionnaire10.txt");
237 
238     Dictionnaire d6 = Dictionnaire_charger("tests/testTADDictionnaire10.txt");
239     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("ça")));
240     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("va")));
241     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("été")));
242     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("sommes")));
243     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("bientôt")));
244     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("élève")));
245     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("noël")));
246     CU_ASSERT_TRUE(Dictionnaire_estPresent(d6, Mot_genererMot("ççdçççéeéé")));
247 
248 }
249 
250 void test_sauvegarder(void) {
251     Dictionnaire d = Dictionnaire_dictionnaire();
252     Mot m = Mot_genererMot("Sal");
253     Dictionnaire_insererMot(&d, m);
254 
255     Fichier f = FichierTexte_fichierTexte("tests/testTADDictionnaire1.txt");
256     FichierTexte_ouvrir(&f, ECRITURE);
257     FichierTexte_ecrireCaractere(&f, '1');
258     FichierTexte_ecrireCaractere(&f, 'S');
259     FichierTexte_ecrireCaractere(&f, '0');
260     FichierTexte_ecrireCaractere(&f, '1');
261     FichierTexte_ecrireCaractere(&f, 'a');
262     FichierTexte_ecrireCaractere(&f, '0');
263     FichierTexte_ecrireCaractere(&f, '1');
264     FichierTexte_ecrireCaractere(&f, 'l');
265     FichierTexte_ecrireCaractere(&f, '1');
266     FichierTexte_ecrireCaractere(&f, '0');
267     FichierTexte_fermer(&f);
268 
269     Dictionnaire_sauvegarder(d, "tests/testTADDictionnaire2.txt");
270     FichierTexte_ouvrir(&f, LECTURE);
271     Fichier f2 = FichierTexte_fichierTexte("tests/testTADDictionnaire2.txt");
272     FichierTexte_ouvrir(&f2, LECTURE);
273     char *ligne1 = FichierTexte_lireChaine(f);
```

```
274     char *ligne2 = FichierTexte_lireChaine(f2);
275
276     CU_ASSERT_TRUE(strcmp(ligne1, ligne2) == 0);
277
278     free(ligne1);
279     free(ligne2);
280
281     FichierTexte_fermer(&f);
282     FichierTexte_fermer(&f2);
283
284     Mot m1 = Mot_genererMot("Salut");
285     Dictionnaire_insererMot(&d, m1);
286     Dictionnaire_sauvegarder(d, "tests/testTADDictionnaire2.txt");
287
288     Fichier f3 = FichierTexte_fichierTexte("tests/testTADDictionnaire4.txt");
289     FichierTexte_ouvrir(&f3, ECRITURE);
290     FichierTexte_ecrireCaractere(&f3, '1');
291     FichierTexte_ecrireCaractere(&f3, 'S');
292     FichierTexte_ecrireCaractere(&f3, '0');
293     FichierTexte_ecrireCaractere(&f3, '1');
294     FichierTexte_ecrireCaractere(&f3, 'a');
295     FichierTexte_ecrireCaractere(&f3, '0');
296     FichierTexte_ecrireCaractere(&f3, '1');
297     FichierTexte_ecrireCaractere(&f3, 'l');
298     FichierTexte_ecrireCaractere(&f3, '1');
299     FichierTexte_ecrireCaractere(&f3, '1');
300     FichierTexte_ecrireCaractere(&f3, 'u');
301     FichierTexte_ecrireCaractere(&f3, '0');
302     FichierTexte_ecrireCaractere(&f3, '1');
303     FichierTexte_ecrireCaractere(&f3, 't');
304     FichierTexte_ecrireCaractere(&f3, '1');
305     FichierTexte_ecrireCaractere(&f3, '0');
306     FichierTexte_fermer(&f3);
307
308     FichierTexte_ouvrir(&f3, LECTURE);
309     Fichier f4 = FichierTexte_fichierTexte("tests/testTADDictionnaire2.txt");
310     FichierTexte_ouvrir(&f4, LECTURE);
311     char *ligne3 = FichierTexte_lireChaine(f3);
312     char *ligne4 = FichierTexte_lireChaine(f4);
313
314     CU_ASSERT_TRUE(strcmp(ligne3, ligne4) == 0);
315
316     free(ligne3);
317     free(ligne4);
318
319     FichierTexte_fermer(&f3);
320     FichierTexte_fermer(&f4);
321
322     Mot m2 = Mot_genererMot("Bonjour");
323     Dictionnaire_insererMot(&d, m2);
```

```
324     Dictionnaire_sauvegarder(d, "tests/testTADDictionnaire2.txt");
325
326     Fichier f5 = FichierTexte_fichierTexte("tests/testTADDictionnaire4.txt");
327     FichierTexte_ouvrir(&f5, ECRITURE);
328     FichierTexte_ecrireCaractere(&f5, '2');
329     FichierTexte_ecrireCaractere(&f5, 'S');
330     FichierTexte_ecrireCaractere(&f5, '0');
331     FichierTexte_ecrireCaractere(&f5, '1');
332     FichierTexte_ecrireCaractere(&f5, 'a');
333     FichierTexte_ecrireCaractere(&f5, '0');
334     FichierTexte_ecrireCaractere(&f5, '1');
335     FichierTexte_ecrireCaractere(&f5, 'l');
336     FichierTexte_ecrireCaractere(&f5, '1');
337     FichierTexte_ecrireCaractere(&f5, '1');
338     FichierTexte_ecrireCaractere(&f5, 'u');
339     FichierTexte_ecrireCaractere(&f5, '0');
340     FichierTexte_ecrireCaractere(&f5, '1');
341     FichierTexte_ecrireCaractere(&f5, 't');
342     FichierTexte_ecrireCaractere(&f5, '1');
343     FichierTexte_ecrireCaractere(&f5, '0');
344     FichierTexte_ecrireCaractere(&f5, 'B');
345     FichierTexte_ecrireCaractere(&f5, '0');
346     FichierTexte_ecrireCaractere(&f5, '1');
347     FichierTexte_ecrireCaractere(&f5, 'o');
348     FichierTexte_ecrireCaractere(&f5, '0');
349     FichierTexte_ecrireCaractere(&f5, '1');
350     FichierTexte_ecrireCaractere(&f5, 'n');
351     FichierTexte_ecrireCaractere(&f5, '0');
352     FichierTexte_ecrireCaractere(&f5, '1');
353     FichierTexte_ecrireCaractere(&f5, 'j');
354     FichierTexte_ecrireCaractere(&f5, '0');
355     FichierTexte_ecrireCaractere(&f5, '1');
356     FichierTexte_ecrireCaractere(&f5, 'o');
357     FichierTexte_ecrireCaractere(&f5, '0');
358     FichierTexte_ecrireCaractere(&f5, '1');
359     FichierTexte_ecrireCaractere(&f5, 'u');
360     FichierTexte_ecrireCaractere(&f5, '0');
361     FichierTexte_ecrireCaractere(&f5, '1');
362     FichierTexte_ecrireCaractere(&f5, 'r');
363     FichierTexte_ecrireCaractere(&f5, '1');
364     FichierTexte_ecrireCaractere(&f5, '0');
365     FichierTexte_fermer(&f5);
366
367     FichierTexte_ouvrir(&f5, LECTURE);
368     Fichier f6 = FichierTexte_fichierTexte("tests/testTADDictionnaire2.txt");
369     FichierTexte_ouvrir(&f6, LECTURE);
370     char *ligne5 = FichierTexte_lireChaine(f5);
371     char *ligne6 = FichierTexte_lireChaine(f6);
372
373     CU_ASSERT_TRUE(strcmp(ligne5, ligne6) == 0);
```

```
374
375     free(ligne5);
376     free(ligne6);
377
378     FichierTexte_fermer(&f5);
379     FichierTexte_fermer(&f6);
380
381     FichierTexte_ouvrir(&f6, LECTURE);
382     CU_ASSERT_TRUE((FichierTexte_lireCaractere(f6) - '0') == d.longueur);
383 }
384
385 void test_estPresent(void) {
386     Dictionnaire d = Dictionnaire_dictionnaire();
387     Mot m = Mot_genererMot("Salut");
388     Mot m2 = Mot_genererMot("Bonjour");
389
390     Dictionnaire_insererMot(&d, m);
391     CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m))
392     CU_ASSERT_FALSE(Dictionnaire_estPresent(d, m2));
393
394     Dictionnaire_insererMot(&d, m2);
395     CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m))
396     CU_ASSERT_TRUE(Dictionnaire_estPresent(d, m2));
397 }
398
399 int main(int argc , char** argv) {
400     CU_pSuite pSuite = NULL;
401
402     /* Initialisation du registre de tests*/
403     if (CUE_SUCCESS != CU_initialize_registry())
404         return CU_get_error() ;
405
406     /* Ajout d'une suite de test */
407     pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
408     if (NULL == pSuite) {
409         CU_cleanup_registry();
410         return CU_get_error();
411     }
412
413     /* Ajout des tests à la suite de tests boite noire */
414     if (
415         (NULL == CU_add_test(pSuite, "test_creationElement", test_creationElement))
416         || (NULL == CU_add_test(pSuite, "Dictionnaire_estVide", test_estVide))
417         || (NULL == CU_add_test(pSuite, "test_insererMot", test_insererMot))
418         || (NULL == CU_add_test(pSuite, "test_insererFichier", test_insererFichier))
419         || (NULL == CU_add_test(pSuite, "test_charger", test_charger))
420         || (NULL == CU_add_test(pSuite, "test_estPresent", test_estPresent))
421         || (NULL == CU_add_test(pSuite, "test_sauvegarder", test_sauvegarder))
422         ) {
423         CU_cleanup_registry() ;
```

```
424         return CU_get_error() ;
425     }
426 
427     /* Lancement des tests */
428     CU_basic_set_mode(CU_BRM_VERBOSE);
429     CU_basic_run_tests();
430     printf ("\n");
431     CU_basic_show_failures( CU_get_failure_list() ) ;
432     printf ("\n\n");
433     /* Nettoyage du registre */
434     CU_cleanup_registry () ;
435     return CU_get_error () ;
436 }
```

## 5.2 ElementDictionnaire

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <CUnit/Basic.h>
4  #include <string.h>
5  #include "TADFichierTexte.h"
6  #include "TADDictionnaire.h"
7  #include "TADElementDictionnairePrive.h"
8  #include "TADMot.h"
9 
10 /*
11 Compilation : make TUdico
12 */
13 
14 int init_suite_success (void) {
15     return 0;
16 }
17 
18 int clean_suite_success (void) {
19     return 0;
20 }
21 
22 void test_element(void) {
23     CU_ASSERT_TRUE(0);
24 }
25 
26 void test_ajouterElementsuivant(void) {
27     char c = 'c';
28     char h = 'h';
29     ElementDictionnaire e1 =  ElementDico_element(c, 1);
30     ElementDictionnaire e2 =  ElementDico_element(h, 1);
31 
32     ElementDico_ajouterElementSuivant(&e1, e2);
33 
```

```
34      CU_ASSERT_EQUAL((*ElementDico_obtenirTableauDesElementsSuivants(e1))[0], e2);
35  }
36
37  void test_supprimerElementsuivant(void) {
38      char c = 'c';
39      char h = 'h';
40      char a = 'a';
41      ElementDictionnaire e1 =  ElementDico_element(c, 1);
42      ElementDictionnaire e2 =  ElementDico_element(h, 1);
43      ElementDictionnaire e3 =  ElementDico_element(a, 1);
44
45      ElementDico_ajouterElementSuivant(&e1, e2);
46      ElementDico_ajouterElementSuivant(&e1, e3);
47      ElementDico_supprimerElementSuivant(&e1, 0);
48
49      CU_ASSERT_EQUAL((*ElementDico_obtenirTableauDesElementsSuivants(e1))[0], e3);
50  }
51
52  void test_ajouterPresence(void) {
53      char c = 'c';
54      ElementDictionnaire e1 =  ElementDico_element(c, 1);
55      CU_ASSERT_TRUE(ElementDico_obtenirBooleen(e1));
56
57      ElementDico_ajouterPresence(&e1, 0);
58      CU_ASSERT_FALSE(ElementDico_obtenirBooleen(e1));
59  }
60
61  void test_obtenirLongueurTableauDesElementsSuivants(void) {
62      char c = 'c';
63      char h = 'h';
64      char a = 'a';
65      char u = 'u';
66      char v = 'v';
67      char e = 'e';
68      ElementDictionnaire e1 =  ElementDico_element(c, 1);
69      ElementDictionnaire e2 =  ElementDico_element(h, 1);
70      ElementDictionnaire e3 =  ElementDico_element(a, 1);
71      ElementDictionnaire e4 =  ElementDico_element(u, 1);
72      ElementDictionnaire e5 =  ElementDico_element(v, 1);
73      ElementDictionnaire e6 =  ElementDico_element(e, 1);
74
75      ElementDico_ajouterElementSuivant(&e1 ,e2);
76      ElementDico_ajouterElementSuivant(&e1 ,e3);
77      ElementDico_ajouterElementSuivant(&e1 ,e4);
78      ElementDico_ajouterElementSuivant(&e1 ,e5);
79      ElementDico_ajouterElementSuivant(&e1 ,e6);
80
81      ElementDico_ajouterElementSuivant(&e3 ,e4);
82      ElementDico_ajouterElementSuivant(&e3 ,e5);
83      ElementDico_ajouterElementSuivant(&e3 ,e6);
```

```
84
85      CU_ASSERT_TRUE(ElementDico_obtenirLongueurTableauDesElementsSuivants(e1) == 5);
86      CU_ASSERT_TRUE(ElementDico_obtenirLongueurTableauDesElementsSuivants(e3) == 3);
87      CU_ASSERT_TRUE(ElementDico_obtenirLongueurTableauDesElementsSuivants(e6) == 0);
88  }
89
90  void test_obtenirLettre(void) {
91      char c = 'c';
92      ElementDictionnaire e1 =  ElementDico_element(c, 1);
93
94      CU_ASSERT_EQUAL(ElementDico_obtenirLettre(e1), c);
95  }
96
97  void test_obtenirBooleen(void) {
98      char c = 'c';
99      ElementDictionnaire e1 =  ElementDico_element(c, 1);
100     ElementDictionnaire e2 =  ElementDico_element(c, 0);
101
102     CU_ASSERT_TRUE(ElementDico_obtenirBooleen(e1));
103     CU_ASSERT_FALSE(ElementDico_obtenirBooleen(e2));
104 }
105
106
107 int main(int argc , char** argv) {
108     CU_pSuite pSuite = NULL;
109
110     /* initialisation du registre de tests*/
111     if (CUE_SUCCESS != CU_initialize_registry())
112         return CU_get_error() ;
113
114     /* ajout d'une suite de test */
115     pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
116     if (NULL == pSuite) {
117         CU_cleanup_registry();
118         return CU_get_error();
119     }
120
121     /* Ajout des tests à la suite de tests boite noire */
122     if (
123         (NULL == CU_add_test(pSuite, "test_ajouterElementsuivant", test_ajouterElementsuivan
124         || (NULL == CU_add_test(pSuite, "test_supprimerElementsuivant", test_supprimerElemen
125         || (NULL == CU_add_test(pSuite, "test_obtenirBooleen", test_obtenirBooleen))
126         || (NULL == CU_add_test(pSuite, "test_obtenirLettre", test_obtenirLettre))
127         || (NULL == CU_add_test(pSuite, "test_obtenirLongueurTableauDesElementsSuivants", te
128         || (NULL == CU_add_test(pSuite, "test_ajouterPresence", test_ajouterPresence))
129         ) {
130         CU_cleanup_registry() ;
131     return CU_get_error() ;
132         }
133
```

```
134     /* Lancement des tests */
135     CU_basic_set_mode(CU_BRM_VERBOSE);
136     CU_basic_run_tests();
137     printf ("\n");
138     CU_basic_show_failures( CU_get_failure_list() ) ;
139     printf ("\n\n");
140     /* Nettoyage du registre */
141     CU_cleanup_registry();
142     return CU_get_error();
143 }
```

## 5.3 FichierTexte

```
1  #include <stdio.h>
2  #include <CUnit/Basic.h>
3  #include <string.h>
4  #include "TADFichierTexte.h"
5
6  /*
7  Compilation : make TUfichier
8  */
9
10 int init_suite_success (void) {
11     return 0;
12 }
13
14 int clean_suite_success (void) {
15     return 0;
16 }
17
18 void test_estOuvert(void){
19     CU_ASSERT_TRUE(FichierTexte_estOuvert(FichierTexte_fichierTexte("tests/testFichierTexte.
20 }
21
22 void test_ouvrir(void){
23     Fichier f1;
24     /* Test pour un fichier ouvert en écriture*/
25     f1 = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
26     FichierTexte_ouvrir(&f1, ECRITURE);
27
28     CU_ASSERT_TRUE(FichierTexte_estOuvert(f1));
29     FichierTexte_fermer(&f1);
30
31
32     /* Test pour un fichier ouvert en lecture*/
33     f1 = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
34     FichierTexte_ouvrir(&f1, LECTURE);
35
36     CU_ASSERT_TRUE(FichierTexte_estOuvert(f1));
```

```
37      FichierTexte_fermer(&f1);
38  }
39
40  void test_mode(void){
41      Fichier f = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
42      FichierTexte_ouvrir(&f, LECTURE);
43
44      CU_ASSERT_EQUAL(FichierTexte_mode(f), LECTURE);
45      FichierTexte_fermer(&f);
46  }
47
48  void test_finFichier(void){
49      Fichier f = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
50      FichierTexte_ouvrir(&f, LECTURE);
51      fseek(f.file, 0, SEEK_END);
52
53      CU_ASSERT_TRUE(FichierTexte_finFichier(f));
54  }
55
56  void test_nonFinFichier(void){
57      Fichier f = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
58      FichierTexte_ouvrir(&f, ECRITURE);
59      FichierTexte_ecrireCaractere(&f, 'a');
60      FichierTexte_fermer(&f);
61      FichierTexte_ouvrir(&f, LECTURE);
62      fseek(f.file, 0, SEEK_SET);
63
64      CU_ASSERT_FALSE(FichierTexte_finFichier(f));
65  }
66
67  void test_lireCaractere(void){
68      Fichier f;
69      char c;
70
71      f = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
72      FichierTexte_ouvrir(&f, ECRITURE);
73      FichierTexte_ecrireCaractere(&f, 'a');
74      FichierTexte_fermer(&f);
75      FichierTexte_ouvrir(&f, LECTURE);
76      fseek(f.file, 0, SEEK_SET);
77      c = FichierTexte_lireCaractere(f);
78
79      CU_ASSERT_EQUAL(c,'a');
80  }
81
82  void test_deplacementCurseurMoinsUn(void){
83      Fichier f;
84
85      f = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
86      FichierTexte_ouvrir(&f, ECRITURE);
```

```
87      FichierTexte_ecrireCaractere(&f, 'a');
88      FichierTexte_fermer(&f);
89      FichierTexte_ouvrir(&f, LECTURE);
90      fseek(f.file, 1, SEEK_SET);
91      FichierTexte_deplacementCurseurMoinsUn(&f);
92
93      CU_ASSERT_EQUAL(ftell(f.file), 0);
94  }
95
96  void test_lireChaine(void){
97      char* chaine;
98      Fichier f;
99      f = FichierTexte_fichierTexte("tests/testFichierTexte.txt");
100     FichierTexte_ouvrir(&f, ECRITURE);
101     FichierTexte_ecrireChaine(&f, "chauve");
102     FichierTexte_fermer(&f);
103
104     FichierTexte_ouvrir(&f, LECTURE);
105     fseek(f.file, 0, SEEK_SET);
106     chaine = FichierTexte_lireChaine(f);
107
108     CU_ASSERT_TRUE(!strcmp(chaine, "chauve"));
109 }
110
111
112 int main(int argc , char** argv){
113     CU_pSuite pSuite = NULL;
114     /* initialisation du registre de tests*/
115     if (CUE_SUCCESS != CU_initialize_registry())
116         return CU_get_error() ;
117
118     /* ajout d'une suite de test */
119     pSuite = CU_add_suite("Tests TAD Fichier Texte ", init_suite_success ,clean_suite_succes
120         if (NULL == pSuite) {
121             CU_cleanup_registry();
122             return CU_get_error();
123         }
124
125     /* Ajout des tests à la suite de tests boite noire */
126     if (
127         (NULL == CU_add_test(pSuite, "FichierTexte_estOuvert", test_estOuvert))
128         || (NULL == CU_add_test(pSuite, "FichierTexte_ouvrir", test_ouvrir))
129         || (NULL == CU_add_test(pSuite, "FichierTexte_mode", test_mode))
130         || (NULL == CU_add_test(pSuite, "FichierTexte_finFichier", test_finFichier))
131         || (NULL == CU_add_test(pSuite, "FichierTexte_nonFinFichier", test_nonFinFichier))
132         || (NULL == CU_add_test(pSuite, "FichierTexte_lireCaractere", test_lireCaractere))
133         || (NULL == CU_add_test(pSuite, "FichierTexte_deplacementCurseurMoinsUn", test_depla
134         || (NULL == CU_add_test(pSuite, "FichierTexte_lireChaine", test_lireChaine))
135         ) {
136         CU_cleanup_registry() ;
```

```
137     return CU_get_error() ;
138         }
139
140     /* Lancement des tests */
141     CU_basic_set_mode(CU_BRM_VERBOSE);
142     CU_basic_run_tests();
143     printf ("\n");
144     CU_basic_show_failures( CU_get_failure_list() ) ;
145     printf ("\n\n");
146     /* Nettoyage du registre */
147     CU_cleanup_registry () ;
148     return CU_get_error () ;
149 }
```

## 5.4 ListeDeMot

```
1  #include <stdio.h>
2  #include <CUnit/Basic.h>
3  #include "TADListeDeMot.h"
4  #include "TADMot.h"
5
6  /*
7  Compilation : make TUliste
8  */
9
10 int init_suite_success (void) {
11     return 0;
12 }
13
14 int clean_suite_success (void) {
15     return 0;
16 }
17
18 void test_estVide(void) {
19     CU_ASSERT_TRUE(ListeDeMot_estVide(ListeDeMot_liste()));
20 }
21
22 void test_ajoutPuisSuppressionAvecUnElement(void) {
23     ListeDeMot liste;
24     Mot mot;
25
26     liste = ListeDeMot_liste();
27     mot = Mot_mot();
28     ListeDeMot_ajouter(&liste, mot);
29
30     CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 1);
31     CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot));
32
33     /* On supprime l'élément */
```

```
34      ListeDeMot_supprimer(&liste, 1);
35      CU_ASSERT_TRUE(ListeDeMot_estVide(ListeDeMot_liste()));
36  }
37
38  void test_multipleInsertionsEtSuppressions(void) {
39      ListeDeMot liste;
40      Mot mot1;
41      Mot mot2;
42      Mot mot3;
43
44      liste = ListeDeMot_liste();
45
46      mot1 = Mot_mot();
47      Mot_insererCaractere(&mot1, 'a', 1);
48      mot2 = Mot_mot();
49      Mot_insererCaractere(&mot2, 'b', 1);
50      mot3 = Mot_mot();
51      Mot_insererCaractere(&mot3, 'c', 1);
52
53      ListeDeMot_ajouter(&liste, mot1);
54      ListeDeMot_ajouter(&liste, mot2);
55      ListeDeMot_ajouter(&liste, mot3);
56
57      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 3);
58      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot1));
59      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot2));
60      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 3), mot3));
61
62      /* On supprime le premier élément */
63      ListeDeMot_supprimer(&liste, 1);
64      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 2);
65      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot2));
66      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot3));
67
68      /* On remet le premier mot au début */
69      ListeDeMot_inserer(&liste, mot1, 1);
70      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 3);
71      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot1));
72      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot2));
73      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 3), mot3));
74
75      /* On supprime le deuxième élément */
76      ListeDeMot_supprimer(&liste, 2);
77      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 2);
78      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot1));
79      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot3));
80
81      /* On remet le deuxième mot à sa place */
82      ListeDeMot_inserer(&liste, mot2, 2);
83      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 3);
```

```
84      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot1));
85      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot2));
86      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 3), mot3));
87
88      /* On supprime le troisième élément */
89      ListeDeMot_supprimer(&liste, 3);
90      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 2);
91      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot1));
92      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot2));
93
94      /* On remet le troisième mot à sa place */
95      ListeDeMot_inserer(&liste, mot3, 3);
96      CU_ASSERT_TRUE(ListeDeMot_longueur(liste) == 3);
97      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot1));
98      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot2));
99      CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 3), mot3));
100
101     /* Suppression de tous les éléments restants */
102     ListeDeMot_supprimer(&liste, 1);
103     CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot2));
104     CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 2), mot3));
105     ListeDeMot_supprimer(&liste, 1);
106     CU_ASSERT_TRUE(Mot_sontEgaux(ListeDeMot_obtenir(liste, 1), mot3));
107     ListeDeMot_supprimer(&liste, 1);
108     CU_ASSERT_TRUE(ListeDeMot_estVide(ListeDeMot_liste()));
109 }
110
111 int main(int argc , char** argv){
112     CU_pSuite pSuite = NULL;
113
114     /* Initialisation du registre de tests */
115     if (CUE_SUCCESS != CU_initialize_registry())
116         return CU_get_error();
117
118     /* Ajout d'une suite de tests */
119     pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
120         if (NULL == pSuite) {
121             CU_cleanup_registry();
122             return CU_get_error();
123         }
124
125
126     /* Ajout des tests à la suite de tests boite noire */
127     if (
128         (NULL == CU_add_test(pSuite, "estVide sur une liste vide", test_estVide))
129         || NULL == CU_add_test(pSuite, "ajout et suppression avec un élement", test_ajoutPui
130         || NULL == CU_add_test(pSuite, "ajouts et suppressions avec plusieurs élements", tes
131         CU_cleanup_registry();
132     return CU_get_error();
133         }
```

```
134
135     /* Lancement des tests */
136     CU_basic_set_mode(CU_BRM_VERBOSE);
137     CU_basic_run_tests();
138     printf ("\n");
139     CU_basic_show_failures(CU_get_failure_list());
140     printf ("\n\n");
141
142     /* Nettoyage du registre */
143     CU_cleanup_registry ();
144
145     return CU_get_error ();
146 }
```

## 5.5 ListeDeSuperMot

```
1  #include <stdio.h>
2  #include <CUnit/Basic.h>
3  #include "TADListeDeSuperMot.h"
4  #include "TADMot.h"
5  #include "TADSuperMot.h"
6
7  /*
8  Compilation : make TUliste
9  */
10
11 int init_suite_success (void) {
12     return 0;
13 }
14
15 int clean_suite_success (void) {
16     return 0;
17 }
18
19 void test_estVide(void) {
20     CU_ASSERT_TRUE(ListeDeSuperMot_estVide(ListeDeSuperMot_liste()));
21 }
22
23 void test_ajoutPuisSuppressionAvecUnElement(void) {
24     ListeDeSuperMot liste;
25     Mot mot;
26     SuperMot supermot;
27
28     liste = ListeDeSuperMot_liste();
29     mot = Mot_mot();
30     SuperMot_fixerMot(&supermot, mot);
31     ListeDeSuperMot_ajouter(&liste, supermot);
32
33     CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 1);
```

```
34      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
35
36      /* On supprime l'élément */
37      ListeDeSuperMot_supprimer(&liste, 1);
38      CU_ASSERT_TRUE(ListeDeSuperMot_estVide(ListeDeSuperMot_liste()));
39  }
40
41  void test_multipleInsertionsEtSuppressions(void) {
42      ListeDeSuperMot liste;
43      Mot mot1;
44      Mot mot2;
45      Mot mot3;
46      SuperMot supermot1;
47      SuperMot supermot2;
48      SuperMot supermot3;
49
50      liste = ListeDeSuperMot_liste();
51
52      mot1 = Mot_mot();
53      Mot_insererCaractere(&mot1, 'a', 1);
54      mot2 = Mot_mot();
55      Mot_insererCaractere(&mot2, 'b', 1);
56      mot3 = Mot_mot();
57      Mot_insererCaractere(&mot3, 'c', 1);
58
59      supermot1 = SuperMot_superMot();
60      SuperMot_fixerMot(&supermot1, mot1);
61      supermot2 = SuperMot_superMot();
62      SuperMot_fixerMot(&supermot2, mot2);
63      supermot3 = SuperMot_superMot();
64      SuperMot_fixerMot(&supermot3, mot3);
65
66      ListeDeSuperMot_ajouter(&liste, supermot1);
67      ListeDeSuperMot_ajouter(&liste, supermot2);
68      ListeDeSuperMot_ajouter(&liste, supermot3);
69
70      CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 3);
71      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
72      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
73      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 3)), mot
74
75      /* On supprime le premier élément */
76      ListeDeSuperMot_supprimer(&liste, 1);
77      CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 2);
78      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
79      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
80
81      /* On remet le premier mot au début */
82      ListeDeSuperMot_inserer(&liste, supermot1, 1);
83      CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 3);
```

```
84      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
85      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
86      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 3)), mot
87
88      /* On supprime le deuxième élément */
89      ListeDeSuperMot_supprimer(&liste, 2);
90      CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 2);
91      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
92      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
93
94      /* On remet le deuxième mot à sa place */
95      ListeDeSuperMot_inserer(&liste, supermot2, 2);
96      CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 3);
97      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
98      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
99      CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 3)), mot
100
101     /* On supprime le troisième élément */
102     ListeDeSuperMot_supprimer(&liste, 3);
103     CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 2);
104     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
105     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
106
107     /* On remet le troisème mot à sa place */
108     ListeDeSuperMot_inserer(&liste, supermot3, 3);
109     CU_ASSERT_TRUE(ListeDeSuperMot_longueur(liste) == 3);
110     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
111     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
112     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 3)), mot
113
114     /* Suppression de tous les éléments restants */
115     ListeDeSuperMot_supprimer(&liste, 1);
116     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
117     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 2)), mot
118     ListeDeSuperMot_supprimer(&liste, 1);
119     CU_ASSERT_TRUE(Mot_sontEgaux(SuperMot_obtenirMot(ListeDeSuperMot_obtenir(liste, 1)), mot
120     ListeDeSuperMot_supprimer(&liste, 1);
121     CU_ASSERT_TRUE(ListeDeSuperMot_estVide(ListeDeSuperMot_liste()));
122 }
123
124 int main(int argc , char** argv){
125     CU_pSuite pSuite = NULL;
126
127     /* Initialisation du registre de tests */
128     if (CUE_SUCCESS != CU_initialize_registry())
129         return CU_get_error();
130
131     /* Ajout d'une suite de tests */
132     pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
133         if (NULL == pSuite) {
```

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


    /* Ajout des tests à la suite de tests boite noire */
    if (
        (NULL == CU_add_test(pSuite, "estVide sur une liste vide", test_estVide))
        || NULL == CU_add_test(pSuite, "ajout et suppression avec un élement", test_ajoutPui
        || NULL == CU_add_test(pSuite, "ajouts et suppressions avec plusieurs élements", tes
        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 ();
}
```

## 5.6 Mot

```
#include <stdio.h>
#include <CUnit/Basic.h>
#include <string.h>
#include "TADMot.h"

/*
Compilation : make TUmot
*/

int init_suite_success (void) {
    return 0;
}

int clean_suite_success (void) {
    return 0;
}

void test_mot(void) {
    Mot mot = Mot_mot();

```

```
21      CU_ASSERT_TRUE(mot.longueur == 0);
22      CU_ASSERT_TRUE(strcmp(mot.contenu, "") == 0);
23  }
24
25  void test_obtenirContenu(void) {
26      Mot mot;
27      mot = Mot_genererMot("Salut");
28
29      CU_ASSERT_TRUE(strcmp(Mot_obtenirContenu(mot), "Salut") == 0);
30  }
31
32  void test_sontEgaux(void) {
33      Mot mot1;
34      mot1 = Mot_genererMot("Salut");
35      Mot mot2;
36      mot2 = Mot_genererMot("Salut");
37
38      CU_ASSERT_TRUE(Mot_sontEgaux(mot1, mot2));
39  }
40
41  void test_longueur(void) {
42      Mot mot;
43      mot = Mot_genererMot("Salut");
44
45      CU_ASSERT_TRUE(Mot_longueur(mot) == 5);
46  }
47
48  void test_modifierCaractere(void) {
49      Mot mot;
50      mot = Mot_genererMot("Salut");
51      Mot_modifierCaractere(&mot, 'p', 3);
52
53      CU_ASSERT_TRUE(strcmp(mot.contenu, "Saput") == 0);
54  }
55
56  void test_insererCaractere(void) {
57      Mot mot;
58      mot = Mot_genererMot("Salut");
59      Mot_insererCaractere(&mot, 'a', 6);
60
61      CU_ASSERT_TRUE(strcmp(mot.contenu, "Saluta") == 0);
62      CU_ASSERT_TRUE(mot.longueur == 6);
63  }
64
65  void test_supprimerCaractere(void) {
66      Mot mot;
67      mot = Mot_genererMot("Salut");
68      Mot_supprimerCaractere(&mot, 3);
69
70      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 1) == 'S');
```

```
71      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 2) == 'a');
72      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 3) == 'u');
73      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 4) == 't');
74      CU_ASSERT_TRUE(Mot_longueur(mot) == 4);
75  }
76
77  void test_iemeCaractere(void) {
78      Mot mot;
79      mot = Mot_genererMot("Salut");
80
81      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 1) == 'S');
82      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 2) == 'a');
83      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 3) == 'l');
84      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 4) == 'u');
85      CU_ASSERT_TRUE(Mot_iemeCaractere(mot, 5) == 't');
86  }
87
88  void test_concatener(void) {
89      Mot mot;
90      Mot mot1 = Mot_genererMot("Salut");
91      Mot mot2 = Mot_genererMot("ations");
92
93      mot = Mot_concatener(mot1, mot2);
94
95      CU_ASSERT_TRUE(strcmp(mot.contenu, "Salutations") == 0);
96      CU_ASSERT_TRUE(mot.longueur == 11);
97  }
98
99  void test_estUnMotValide(void) {
100     Mot mot;
101     mot = Mot_genererMot("Bl-ah");
102     CU_ASSERT_TRUE(Mot_estUnMotValide(mot));
103 }
104
105 void test_separerMots(void) {
106     Mot mot = Mot_genererMot("Salut");
107     Mot mot1 = Mot_mot();
108     Mot mot2 = Mot_mot();
109
110     Mot_separerMots(mot, 3, &mot1, &mot2);
111     CU_ASSERT_TRUE(strcmp(mot1.contenu, "Sa") == 0);
112     CU_ASSERT_TRUE(strcmp(mot2.contenu, "lut") == 0);
113     Mot_separerMots(mot, 2, &mot1, &mot2);
114     CU_ASSERT_TRUE(strcmp(mot1.contenu, "S") == 0);
115     CU_ASSERT_TRUE(strcmp(mot2.contenu, "alut") == 0);
116     Mot_separerMots(mot, 5, &mot1, &mot2);
117     CU_ASSERT_TRUE(strcmp(mot1.contenu, "Salu") == 0);
118     CU_ASSERT_TRUE(strcmp(mot2.contenu, "t") == 0);
119 }
120
```

```
121 void test_sousMot(void) {
122     Mot mot;
123     mot = Mot_genererMot("Casse-croute");
124 
125     CU_ASSERT_TRUE(strcmp(Mot_sousMot(mot, 7, 6).contenu, "croute") == 0);
126     CU_ASSERT_TRUE(strcmp(Mot_sousMot(mot, 1, 1).contenu, "C") == 0);
127     CU_ASSERT_TRUE(strcmp(Mot_sousMot(mot, 1, Mot_longueur(mot)).contenu, "Casse-croute") ==
128     CU_ASSERT_TRUE(strcmp(Mot_sousMot(mot, 1, 5).contenu, "Casse") == 0);
129 }
130 
131 void test_estSousMot(void) {
132     Mot mot1;
133     mot1 = Mot_genererMot("Casse-croute");
134     Mot mot2;
135     mot2 = Mot_genererMot("croute");
136     CU_ASSERT_TRUE(Mot_estSousMot(mot2, mot1));
137 }
138 
139 void test_sontIdentiques(void) {
140     Mot mot1;
141     mot1 = Mot_genererMot("casse");
142     Mot mot2;
143     mot2 = Mot_genererMot("casse");
144     CU_ASSERT_FALSE(Mot_sontIdentiques(&mot1, &mot2));
145     CU_ASSERT_TRUE(Mot_sontIdentiques(&mot1, &mot1));
146 }
147 
148 
149 int main(int argc , char** argv){
150     CU_pSuite pSuite = NULL;
151 
152     /* initialisation du registre de tests*/
153     if (CUE_SUCCESS != CU_initialize_registry())
154         return CU_get_error() ;
155 
156     /* ajout d'une suite de test */
157     pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
158         if (NULL == pSuite) {
159             CU_cleanup_registry();
160             return CU_get_error();
161         }
162 
163 
164     /* Ajout des tests à la suite de tests boite noire */
165     if (
166         (NULL == CU_add_test(pSuite, "Mot_mot", test_mot))
167         || (NULL == CU_add_test(pSuite, "Mot_obtenirContenu", test_obtenirContenu))
168         || (NULL == CU_add_test(pSuite, "Mot_sontEgaux", test_sontEgaux))
169         || (NULL == CU_add_test(pSuite, "Mot_longueur", test_longueur))
170         || (NULL == CU_add_test(pSuite, "Mot_modifierCaractere", test_modifierCaractere))
```

```
171         || (NULL == CU_add_test(pSuite, "Mot_insererCaractere", test_insererCaractere))
172         || (NULL == CU_add_test(pSuite, "Mot_supprimerCaractere", test_supprimerCaractere))
173         || (NULL == CU_add_test(pSuite, "Mot_iemeCaractere", test_iemeCaractere))
174         || (NULL == CU_add_test(pSuite, "Mot_concatener", test_concatener))
175         || (NULL == CU_add_test(pSuite, "Mot_estUnMotValide", test_estUnMotValide))
176         || (NULL == CU_add_test(pSuite, "Mot_separerMots", test_separerMots))
177         || (NULL == CU_add_test(pSuite, "Mot_sousMot", test_sousMot))
178         || (NULL == CU_add_test(pSuite, "Mot_estSousMot", test_estSousMot))
179         || (NULL == CU_add_test(pSuite, "Mot_sontIdentiques", test_sontIdentiques))
180         ) {
181         CU_cleanup_registry() ;
182     return CU_get_error() ;
183         }
184 
185     /* Lancement des tests */
186     CU_basic_set_mode(CU_BRM_VERBOSE);
187     CU_basic_run_tests();
188     printf ("\n");
189     CU_basic_show_failures( CU_get_failure_list() ) ;
190     printf ("\n\n");
191     /* Nettoyage du registre */
192     CU_cleanup_registry () ;
193     return CU_get_error () ;
194 }
```

## 5.7 Parametres

```
1  #include <stdio.h>
2  #include <CUnit/Basic.h>
3  #include <string.h>
4  #include "TADParametres.h"
5  
6  /*
7  Compilation : make TUparametres
8  */
9  
10 int init_suite_success (void) {
11     return 0;
12 }
13 
14 int clean_suite_success (void) {
15     return 0;
16 }
17 
18 void test_fixerDictionnaire(void) {
19     Parametres parametre = Parametres_parametre();
20 
21     Parametres_fixerDictionnaire(&parametre, "Dico.txt");
22     CU_ASSERT_STRING_EQUAL(Parametres_obtenirDictionnaire(parametre), "Dico.txt");
```

```
23 }
24
25 void test_fixerFichier(void) {
26     Parametres parametre = Parametres_parametre();
27
28     Parametres_fixerFichier(&parametre, "Fichier.txt");
29     CU_ASSERT_STRING_EQUAL(Parametres_obtenirFichier(parametre), "Fichier.txt");
30 }
31
32 void test_fixerCible(void) {
33     Parametres parametre = Parametres_parametre();
34
35     Parametres_fixerCible(&parametre, "Cible.txt");
36     CU_ASSERT_STRING_EQUAL(Parametres_obtenirCible(parametre), "Cible.txt");
37 }
38
39 void test_fixerAide(void) {
40     Parametres parametre = Parametres_parametre();
41
42     Parametres_fixerAide(&parametre, 1);
43     CU_ASSERT_EQUAL(Parametres_obtenirAide(parametre), 1);
44 }
45
46 int main(int argc , char** argv){
47     CU_pSuite pSuite = NULL;
48
49     /* initialisation du registre de tests */
50     if (CUE_SUCCESS != CU_initialize_registry())
51         return CU_get_error() ;
52
53     /* ajout d'une suite de test */
54     pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
55     if (NULL == pSuite) {
56         CU_cleanup_registry();
57         return CU_get_error();
58     }
59
60     /* Ajout des tests à la suite de tests boite noire */
61     if (
62         (NULL == CU_add_test(pSuite, "Fixer Dico", test_fixerDictionnaire))
63         || (NULL == CU_add_test(pSuite, "Fixer Fichier", test_fixerFichier))
64         || (NULL == CU_add_test(pSuite, "Fixer Cible", test_fixerCible))
65         || (NULL == CU_add_test(pSuite, "Fixer Aide", test_fixerAide))
66         ) {
67         CU_cleanup_registry() ;
68         return CU_get_error() ;
69     }
70
71     /* Lancement des tests */
72     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 () ;
}
```

## 5.8 SuperMot

```
#include <stdio.h>
#include <CUnit/Basic.h>
#include <string.h>
#include "TADMot.h"
#include "TADListeDeMot.h"
#include "TADSuperMot.h"

/*
Compilation : make TUsupermot
*/

int init_suite_success (void) {
    return 0;
}

int clean_suite_success (void) {
    return 0;
}

void test_fixerMots(void) {
    SuperMot superMot = SuperMot_superMot();
    Mot mot = Mot_genererMot("Salut");

    SuperMot_fixerMot(&superMot, mot);

    CU_ASSERT_STRING_EQUAL(Mot_obtenirContenu(SuperMot_obtenirMot(superMot)), "Salut");
    CU_ASSERT_STRING_NOT_EQUAL(Mot_obtenirContenu(SuperMot_obtenirMot(superMot)), "Blah");
}

void test_position(void) {
    SuperMot superMot = SuperMot_superMot();

    CU_ASSERT_TRUE(SuperMot_obtenirPosition(superMot) == 1);

    SuperMot_fixerPosition(&superMot, 42);

    CU_ASSERT_FALSE(SuperMot_obtenirPosition(superMot) == 1);
    CU_ASSERT_TRUE(SuperMot_obtenirPosition(superMot) == 42);
```

```
39  }
40
41  void test_listeCorrection(void) {
42
43      SuperMot superMot = SuperMot_superMot();
44      CU_ASSERT_TRUE(ListeDeMot_estVide(SuperMot_obtenirListeDeCorrection(superMot)));
45
46      ListeDeMot liste = ListeDeMot_liste();
47      Mot mot = Mot_genererMot("Salut");
48      ListeDeMot_ajouter(&liste, mot);
49      SuperMot_fixerListeDeCorrections(&superMot, liste);
50      CU_ASSERT_FALSE(ListeDeMot_estVide(SuperMot_obtenirListeDeCorrection(superMot)));
51
52      SuperMot_fixerListeDeCorrections(&superMot, ListeDeMot_liste());
53      CU_ASSERT_TRUE(ListeDeMot_estVide(SuperMot_obtenirListeDeCorrection(superMot)));
54  }
55
56  void test_validite(void) {
57      SuperMot superMot = SuperMot_superMot();
58
59      CU_ASSERT_FALSE(SuperMot_obtenirValidite(superMot));
60
61      SuperMot_fixerValidite(&superMot, 1);
62
63      CU_ASSERT_TRUE(SuperMot_obtenirValidite(superMot));
64
65      SuperMot_fixerValidite(&superMot, 0);
66
67      CU_ASSERT_FALSE(SuperMot_obtenirValidite(superMot));
68  }
69
70  int main(int argc , char** argv){
71      CU_pSuite pSuite = NULL;
72
73      /* initialisation du registre de tests*/
74      if (CUE_SUCCESS != CU_initialize_registry())
75          return CU_get_error() ;
76
77      /* ajout d'une suite de test */
78      pSuite = CU_add_suite("Tests boite noire ", init_suite_success ,clean_suite_success );
79          if (NULL == pSuite) {
80              CU_cleanup_registry();
81              return CU_get_error();
82          }
83
84
85      /* Ajout des tests à la suite de tests boite noire */
86      if (
87          (NULL == CU_add_test(pSuite, "SuperMot_fixerMot", test_fixerMots))
88          || (NULL == CU_add_test(pSuite, "SuperMot_position", test_position))
```

```
89          || (NULL == CU_add_test(pSuite, "SuperMot_listeCorrection", test_listeCorrection))
90          || (NULL == CU_add_test(pSuite, "SuperMot_validite", test_validite))
91          ) {
92          CU_cleanup_registry() ;
93      return CU_get_error() ;
94          }
95
96      /* Lancement des tests */
97      CU_basic_set_mode(CU_BRM_VERBOSE);
98      CU_basic_run_tests();
99      printf ("\n");
100     CU_basic_show_failures( CU_get_failure_list() ) ;
101     printf ("\n\n");
102     /* Nettoyage du registre */
103     CU_cleanup_registry () ;
104     return CU_get_error () ;
105 }
```

# Cinquième partie

# Répartition du travail et conclusion

# Chapitre 1

# Conclusions personelles

**Sommaire**



## 1.1 Antoine Augusti

La réalisation de ce projet algorithmique m'a permis avant tout de travailler avec un langage qui n'a pas de *ramasse-miettes* au niveau de la mémoire. La gestion de la mémoire dynamique est quelque chose de complexe et j'ai trouvé la réalisation de ce projet très intéressante surtout parce que le C est un langage très sévère au niveau de la gestion de la mémoire. J'ai été également très satisfait de l'ensemble du travail de notre équipe car nous avons su tenir les délais imposés, sans chercher à avancer trop vite et sans prendre de retard.

En revanche, j'étais déjà habitué à travailler sur des projets à plusieurs développeurs, avec des gestionnaires de versions et une répartition du travail entre les différentes personnes où la production d'une documentation complète est primordiale. Ce projet ne m'a donc rien appris de nouveau de ce côté car ce sont des méthodes de travail que je connais et pratique depuis plusieurs années.

## 1.2 Étienne Batise

Ce projet m'a permis tout d'abord d'apprendre à développer dans un langage rigoureux et peu permissif : le C. Il m'a permis de mieux comprendre les étapes de la compilation aussi bien que les spécificité de la gestion de la mémoire.

Étant habitué à travailler sur des projets personnels en groupe, travailler avec un cahier de charges précis m'a permis d'améliorer mes capacités à m'adapter à différentes contraintes comme un calendrier prévisionnel ou des logiciels précis.

## 1.3 Jean-Claude Bernard

## 1.4 Thibaud Dauce

## 1.5 Faustine Demiselle

# Chapitre 2

# Conclusion générale

**Sommaire**



## 2.1 Répartition des tâches

D'un point de vue de l'organisation, nous avons d'abord fait l'analyse ensemble afin de se mettre d'accord sur le rôle de chaque TAD. Puis nous avons réparti les tâches au fur et à mesure du projet, en essayant de ne pas avoir la même personne sur la même partie deux fois de suite afin de mieux détecter les erreurs. Voici comment le travail a été réparti (JC : Jean-Claude. On écrit JC parce que sinon Jean-Claude ça prend trop de place dans le tableau.) :

| | TAD / CP | Conception détaillée | Dév | TU |
|---|---|---|---|---|
| Mot | Étienne et Faustine | Antoine | Thibaud | JC |
| SuperMot | Antoine | Antoine | Faustine | Thibaud |
| Dictionnaire | Thibaud et JC | Thibaud | Antoine et JC | Étienne |
| ElementDictionnaire | Thibaud et JC | JC | Antoine et JC | Faustine |
| CorrecteurOrthographique | Faustine | Faustine | Étienne | Antoine |
| Paramètres | Étienne | Étienne | Faustine | Thibaud |
| Liste | M. Delestre | M. Delestre | Étienne et Thibaud | Antoine |
| FichierTexte | M. Delestre | Étienne | Étienne | Étienne |
| Logique métier | Tous | Tous | Tous | Tous |

Les fonctions de logique métier étant nombreuses et délicates en raison de la gestion de l'ensemble des TAD, nous les avons gardées pour la fin et les avons finalement partagées, c'est pourquoi tout le monde a travaillé dessus comme indiqué dans le tableau.

## 2.2 Respect du planning

Retrouvez dans le tableau ci-dessous le planning d'avancement prévu par M. Delestre et l'avancement réel de la réalisation de notre projet.

| Date | Tâche prévue | Tâche réalisée |
|---|---|---|
| 6 novembre | TAD et rapport | TAD et rapport |
| 13 novembre | Analyse descendante | Analyse descendante |
| 20 novembre | CP et rapport | CP et rapport |
| 27 novembre | CD et rapport | CD et rapport |
| 4 décembre | Développement et TU | CD et rapport |
| 11 décembre | Développement et TU | Développement et TU |
| 18 décembre | Développement et TU | Développement et TU |
| Vacances de Noël | Développement et TU | Débogage et rapport |

## 2.3 La véritable vraie conclusion

Pour conclure, on peut dire que ce projet est pour nous un succès tant au niveau de la réalisation que de l'organisation.

D'un point de vue du travail d'équipe, nous sommes également satisfaits. Même si nous commençons à être habitués au travail d'équipe, c'était pour nous le premier véritable projet informatique d'une ampleur assez importante.

Ce projet nous a permis d'apprendre comment travailler correctement en équipe sur un projet informatique, c'est-à-dire comment bien répartir le travail et comment bien s'organiser, par exemple en faisant en groupe l'analyse descendante et la conception préliminaire. Nous avons compris l'importance qu'avaient les .h et la documentation pour que tout le monde puisse savoir comment utiliser toutes les fonctions. Ainsi, nous respectons le principe d'encapsulation que l'on sait maintenant quasi primordial pour de tels projets.

Nous avons également appris à utiliser de nouveaux outils adaptés à ce type de projet comme un dépôt SVN ainsi que le framework CUnit qui nous a permis de faire tous nos tests unitaires plus simplement.

Enfin, ce projet aura été notre première pratique importante du C, nous ayant appris à manipuler correctement ce langage et commencer a maîtriser certaines de ses particularités et différences avec l'algorithmique.