INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE

ASIspell

logastar.com

Dounia BOUTAYEB
Thibault SAURON
Matthias SESBOÜÉ
Damien TOOMEY

ASIspell

Projet Correcteur Orthographique

Algorithmique avancée et programmation C - ASI - 2017-2018

Table des matières

1 Introduction **3**

2 Résumé des séances **4**
2.1 Déroulement du projet . . . 4
2.2 Choix des conventions de nommage dans les .c et .h . . . 5

3 Choix de la structure de donnée du Dictionnaire **7**
3.1 Choix possibles . . . 7
3.1.1 Tableau . . . 7
3.1.2 Arbre binaire de recherche . . . 7
3.1.3 Arbre n-aire . . . 8
3.1.4 Choix final . . . 8

4 Explication du Type Dictionnaire **9**

5 Analyse **11**
5.1 Types Abstraits de Données (TAD) . . . 11
5.1.1 TAD Mot . . . 11
5.1.2 TAD Dictionnaire . . . 11
5.1.3 TAD CorrecteurOrthographique . . . 12
5.1.4 TAD Arbre n-aire . . . 12
5.2 Analyse descendante . . . 14

6 Conception préliminaire **19**
6.1 Signatures liées au TAD Mot . . . 19
6.2 Signatures liées au TAD Dictionnaire . . . 19
6.3 Signatures liées au TAD CorrecteurOrthographique . . . 20
6.4 Signatures liées au TAD Arbre n-aire . . . 20
6.5 Signatures des procédures d'affichage . . . 21
6.6 Signatures des procédures privées du main . . . 21
6.7 Signatures liées au TAD Fichier Texte donné dans le sujet . . . 21

7 Conception détaillée **23**
7.1 Fonctions et procédures liées au TAD Mot . . . 23
7.1.1 Partie Privée . . . 23
7.1.2 Partie Publique . . . 23
7.2 Fonctions et procédures liées au TAD Dictionnaire . . . 26
7.2.1 Partie Privée . . . 26
7.2.2 Partie publique . . . 28
7.3 Fonctions et procédures liées au TAD CorrecteurOrthographique . . . 30
7.3.1 Partie privée . . . 30

7.3.2 Partie Publique 31
7.4 Fonctions et procédures liées au TAD ArbreN 33
7.5 Procédures d'affichage 35
7.5.1 Partie privée 35
7.5.2 Partie publique 35
7.6 Procédures privées du main 36

8 Code C **38**
8.1 mot.h 38
8.2 mot.c 39
8.3 dictionnaire.h 42
8.4 dictionnaire.c 43
8.5 correcteurOrthographique.h 48
8.6 correcteurOrthographique.c 49
8.7 arbreN.h 52
8.8 arbreN.c 54
8.9 ensemble.h 55
8.10 ensemble.c 56
8.11 listeChaineeMot.h 58
8.12 listeChaineeMot.c 60
8.13 affichages.h 61
8.14 affichages.c 62
8.15 existeFichier.h 63
8.16 existeFichier.c 63
8.17 main.c 64

9 Tests unitaires **68**
9.1 motTU.c 68
9.2 dictionnaireTU.c 76
9.3 correcteurOrthographiqueTU.c 82
9.4 arbreNTU.c 90
9.5 ensembleTU.c 96
9.6 listeChaineeMotTU.c 103

10 Conclusion **110**
10.1 Conclusion générale 110
10.2 Conclusions personnelles 110
10.2.1 Dounia BOUTAYEB 110
10.2.2 Thibault SAURON 110
10.2.3 Matthias SESBOÜÉ 110
10.2.4 Damien TOOMEY 111

11 Répartition du travail **112**
11.1 Répartition générale 112
11.2 Travaux communs 112

1 Introduction

L'objectif de ce projet est de développer un correcteur orthographique efficace à l'image des programmes Unix ISPELL et Aspell. Le programme doit d'une part, pouvoir analyser un texte qui lui est donné via l'entrée standard et proposer des corrections orthographiques si besoin. Cette analyse est dépendante d'un dictionnaire qui contient au départ 336531 mots. Le programme doit d'autre part, donner la possibilité de compléter un dictionnaire à l'aide des mots d'un fichier texte (un mot par ligne).

Nous avons été réparti en groupe de quatre étudiants. Ce projet permet de nous mettre dans des conditions d'entreprise avec un chef de projet, un délai et un cahier des charges à respecter. Le langage de programmation imposé est le C. Nous devons utiliser la plateforme monprojet pour faciliter l'échange du code.

2 Résumé des séances

2.1 Déroulement du projet

25/10/2017 : Découverte du sujet et du groupe.
Création du projet sur la plateforme `https://monprojet.insa-rouen.fr`.
Conception de la première version des TAD Mot, Dictionnaire et CorrecteurOrthograpique.

08/11/2017 : Travail sur l'analyse descendante.

15/11/2017 : Remise en cause des TAD.
Suppression de plusieurs fonctions et procédures non nécessaires dans le cadre de notre projet.
Ajouts et modifications de plusieurs fonctions et procédures.
Le TAD Mot contient les opérations permettant de faire toutes les modifications possibles sur un mot.
Le TAD CorrecteurOrhtographique contient toutes les opérations de correction. Ces opérations sont en accord avec le fait que l'on traitera le texte à corriger comme un flux. On parcours le texte et on corrige mot par mot. On ne traite donc plus une liste de mots incorrectes. Choix de la structure de donnée du type Dictionnaire.
On choisit la structure de donnée dynamique arbre N-aire, que l'on stockera à la manière d'un arbre binaire.

22/11/2017 : Remise en cause de l'analyse descendante.
Re-manipulation de l'analyse descendante, ajout de toutes les fonctions et procédures des TAD et suppression des procédures d'affichage.
Séparation du travail pour la conception détaillée :

- *Damien* travaille sur les fonctions et procédures "queFaireEnFonctionDeCommandeDEntree", "afficherAide", ainsi que les fonctions et procédures du TAD "CorrecteurOrthographique".
- *Dounia* travaille sur les fonctions et procédures du TAD Mot.
- *Thibault* travaille sur les fonctions et procédures du TAD Dictionnaire.
- *Matthias* travaille avec *Damien* sur les fonctions et procédures du TAD CorrecteurOrthographique, ainsi que sur le procédures "afficherMotBienEcrit", "afficherCorrectionsEtPosMot", "correctionDuMot" et "corrigerMot".

29/11/2017 : Analyse descendante finalisée, conception détaillée en cours. Séparation du travail d'implémentation, à faire dès que la conception détaillée est terminée.
Séparation du travail pour la conception détaillée :

- *Damien* développe les tests unitaires pour le TAD Mot (sauf celui de la fonction "obtenirMotEntreeStandard").
- *Dounia* développe les fonctions et procédures du TAD Ensemble.
- *Thibault* développe les tests unitaires pour le TAD CorrecteurOrhtographique ainsi que les fonctions et procédures du TAD Dictionnaire.
- *Matthias* développe les tests unitaires pour le TAD Dictionnaire et celui pour la fonction "obtenirMotEntreeStandard" ainsi que les fonctions et procédures du TAD CorrecteurOrthographique.

06/12/2017 : Nous continuons à développer en C et ajout de quelques modifications.

Modifications apportées :

. Définition du SDD pour le type CorrecteurOrthographique : une structure contenant un Mot et un Dictionnaire. En conséquence nous avons ajouté des fonctions accesseurs pour obtenir le Dictionnaire et le Mot.

. Définition du SDD pour le type Ensemble : une structure contenant deux champs, un pour les éléments et un autre pour le nombre d'éléments.

. Nous avons choisi de développer notre propre type ListeChaineeMot étant donné que nous n'utilisons qu'une liste chaînée de Mot et que nous n'avons besoin que de certaines fonctions et procédures.

Séparation du travail :

. *Damien* et *Thibault* développent ensemble les TAD ArbreNaire et Dictionnaire.

. *Dounia* développe les TAD ListeChaineeMot et Mot.

. *Matthias* développe le TAD CorrecteurOrthographique.

13/12/2017 : Nous continuons à développer en C. Suite à une mauvaise organisation de départ, nous devons vérifier les conventions de nommage pour nos codes soient cohérents entre eux.

Séparation du travail :

. *Damien* et *Thibault* développent ensemble les procédures de sérialisation et de désérialisation.

. *Dounia* développe des tests unitaires.

. *Matthias* vérifie que les conventions de nommage sont bien respectées et travaille ensuite sur les tests unitaires.

20/12/2017 : Le développement de base est terminé, le code général ainsi que les tests unitaires sont développés. La séance consistait donc à faire passer les tests unitaires et donc commencer à corriger les code.

Cette séance était la dernière. L'objectif pour la fin de la semaine est de finir le code.

2.2 Choix des conventions de nommage dans les .c et .h

Une fois la conception détaillée terminée, nous avons dû choisir des conventions de nommage pour pouvoir développer notre partie du code chacun de notre côté tout en restant cohérent avec le code des autres. Nous avons donc choisie les conventions suivantes :

TAD Mot : Nous adopterons les conventions suivantes :

. Le type Mot sera nommé : *M_ Mot*.

. Les fonctions et procédures du TAD Mot seront nommées de la façon suivante : *M_ nomDeLaFonction ou M_ nomDeLaProcédure*.

TAD Dictionnaire : Nous adopterons les conventions suivantes :

. Le type Dictionnaire sera nommé : *D_ Dictionnaire*.

. Les fonctions et procédures du TAD Dictionnaire seront nommées de la façon suivante : *D_ nomDeLaFonction ou D_ nomDeLaProcédure*.

TAD CorrecteurOrthographique : Nous adopterons les conventions suivantes :

. Le type CorrecteurOrthographique sera nommé : *CO_ CorrecteurOrthographique*.

. Les fonctions et procédures du TAD CorrecteurOrthographique seront nommées de la façon suivante :
CO_nomDeLaFonction ou CO_nomDeLaProcédure.

TAD Ensemble : Nous adopterons les conventions suivantes :

. Le type Ensemble sera nommé : *E_Ensemble.*

. Les fonctions et procédures du TAD Ensemble seront nommées de la façon suivante :
E_nomDeLaFonction ou E_nomDeLaProcédure.

TAD ListeChaineeMot : Nous adopterons les conventions suivantes :

. Le type ListeChaineeMot sera nommé : *LCM_ListeChaineeMot.*

. Les fonctions et procédures du TAD ListeChaineeMot seront nommées de la façon suivante :
LCM_nomDeLaFonction ou LCM_nomDeLaProcédure.

TAD ArbreN : Nous adopterons les conventions suivantes :

. Le type ArbreN sera nommé : *AbN_ArbreN.*

. Les fonctions et procédures du TAD ArbreN seront nommées de la façon suivante :
AbN_nomDeLaFonction ou Abn_nomDeLaProcédure.

3 Choix de la structure de donnée du Dictionnaire

3.1 Choix possibles

Nous allons ici discuter de trois choix possibles que nous avons pour stocker les mots contenus dans le dictionnaire. Nous verrons les avantages et les inconvénients de ces différentes structures de donnée et nous finirons par expliquer notre choix final.

3.1.1 Tableau

Nous pourrions, très naïvement, choisir de stocker l'ensemble des mots du dictionnaire dans un tableau.
Cette structure de donnée statique aurait pour seul avantage un accès aux mots en $\mathcal{O}(1)$.
En revanche pour ce qui est de ses désavantages ils sont nombreux :

- Un tableau est une structure de donnée statique. Or nous devons pouvoir insérer des mots facilement et donc faire varier la taille de ce tableau. Dans le cas présent d'une structure de donnée statique, nous devons choisir une taille maximum pour notre tableau à la compilation, ce qui n'est absolument pas intéressant.
- L'insertion d'un nouveau mot va aussi poser problème puisqu'il faudra à chaque fois faire un décalage de tous les mots qui sont situés après l'indice d'insertion, avant d'insérer le mot.
- Un autre problème réside dans la recherche d'un mot. En effet dans le cas d'un tableau, la recherche naïve d'un mot sera en $\mathcal{O}(\mathrm{n})$ où n correspond au nombre de mot du dictionnaire, qui sera donc très grand. Cette recherche ne sera par conséquent absolument pas efficace. On pourrait l'améliorer en faisant une recherche par dichotomie mais le fait que nous souhaitons stocker des mots rend sont utilisation impossible dans notre cas.

3.1.2 Arbre binaire de recherche

Nous pourrions choisir de stocker l'ensemble des mots à l'aide d'un arbre binaire de recherche.
Cette structure de donnée dynamique présente deux gros avantages :

- Un arbre binaire de recherche présente des caractéristiques particulières qui permettent la recherche d'un élément en $\mathcal{O}(\log(\mathrm{n}))$ ce qui est très intéressant dans notre cas puisque nous allons devoir rechercher dans le dictionnaire si un mot existe, et ce avec une contrainte de temps.
- Cette structure de donnée est dynamique. Ceci implique que nous pourrions faire varier la taille de notre arbre selon nos besoins au cours de l'exécution du programme, ce qui est un avantage important qui répond à un besoin de notre sujet.

Elle nécessiterait en revanche de quantifier chaque mot avec une valeur unique, ce qui reste faisable mais compliqué, et de mettre en place un moyen d'équilibrer l'arbre afin d'avoir des temps de recherche correct.

3.1.3 Arbre n-aire

Enfin, nous pourrions choisir de stocker nos mots grâce à un arbre n-aire.

. Cette structure de donnée est aussi dynamique, en découle le même avantage que pour l'arbre binaire de recherche.
. L'arbre n-aire permettrait de stocker des lettres et non pas des mots. Ainsi, un mot devient un chemin de parcours de l'arbre. Par conséquent la recherche d'un mot se fait en $\mathcal{O}(n)$ où n est le nombre de lettres maximal des mots. Ce qui donnerai un temps de recherche acceptable.
. Un avantage de cet arbre est la place qu'occupe la structure : comme les mots sont représentés par un chemin, on peut stocker par exemple 'abc' et 'ab' avec seulement 3 noeuds, ce qui permet d'avoir une structure de donnée nécessitant un espace mémoire relativement faible.

3.1.4 Choix final

Pour les raisons d'optimisation présentées précédemment, nous ne choisirons pas un tableau. Pour ce qui est des arbres le choix est plus compliqué. Les deux types d'arbres sont intéressant et le choix s'est fait sur la complexité des algorithmes à développer. En effet dans le cas d'un arbre n-aire et dans celui d'un arbre binaire de recherche le temps moyen de recherche est environ le même. Dans l'arbre binaire de recherche, la hauteur d'un arbre est fonction du nombre de mots alors que pour l'arbre n-aire elle est fonction du nombre de lettres des mots. La hauteur maximum de celui-ci sera donc le nombre de lettres du plus grand mot. Ce qui nous donne pour un arbre binaire de recherche une taille de $\log_2(300000) = 18.19$, soit une hauteur maximale de 19, avec une partie importante des mots qui sera située sur les feuilles de par la construction de l'arbre.
Pour l'arbre n-aire, la taille maximale est plus grande (anticonstitutionnellement fait 25 lettres) mais en moyenne en France les mots font entre 6 et 7 caractères suivant les sources, donc la plupart des mots seront situés dans une hauteur de 7 environ. De plus nous n'avons pas besoin de parcourir tout le tableau pour savoir si un mot est mal orthographié.
Là où l'arbre binaire de recherche devient un peu moins intéressant c'est lorsqu'il s'agit d'ajouter (ou de supprimer) un mot dans le dictionnaire. Dans la mesure où, pour assurer la complexité de recherche d'un élément en $\mathcal{O}(\log(n))$, il faut s'assurer que l'arbre reste équilibré en faisant des rotations après chaque insertion, cela implique des algorithmes plus compliqués ; en tout cas plus compliqués que pour l'arbre n-aire. Les performances des deux arbres ayant l'air à peu près équivalente nous choisirons celui qui sera plus simple à réaliser : l'arbre n-aire.

4 Explication du Type Dictionnaire

Notre dictionnaire est donc un arbre n-aire. Nous allons détailler son utilisation.
Un arbre n-aire peut être visualisé comme un arbre binaire mais où seulement un seul fils fait 'descendre' dans la profondeur de l'arbre, on parle de fils et de frère. comme dans l'exemple qui suit :

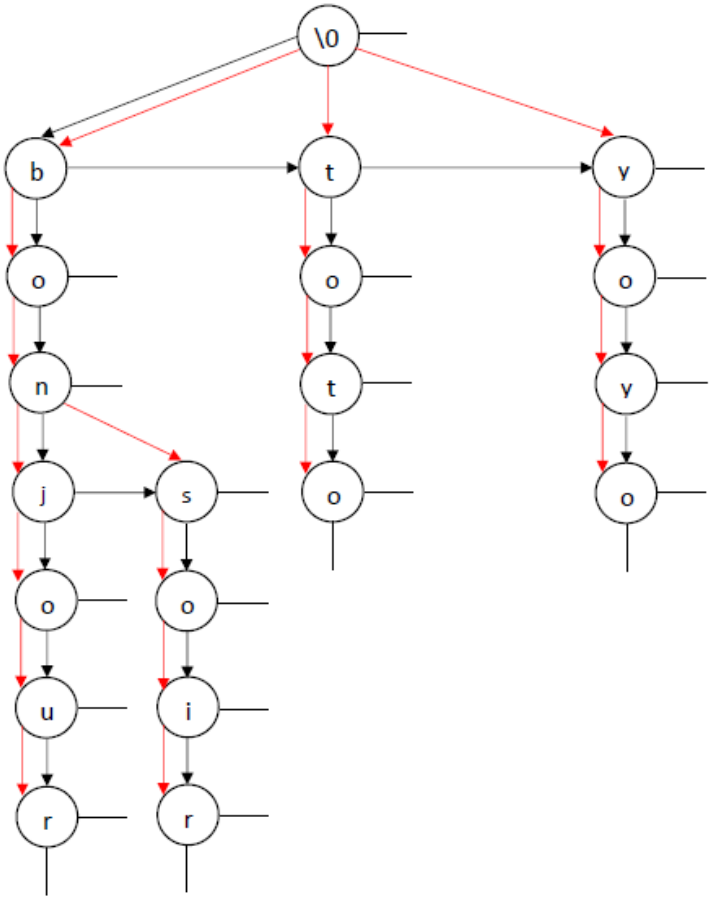


FIGURE 1 – Un exemple d'arbre n-aire

Légende :
Rouge : TAD Collection Arbre
Noir : Conception à l'aide d'un Arbre Binaire

Dans notre dictionnaire, la racine n'est pas utilisée. Elle est ici présente car, par définition, la racine d'un arbre n-aire a un frère vide du point de vue de la conception. Ici le 'a' est le fils de la racine et le 't' est le frère de 'a'. L'intérêt de cette représentation est qu'un mot devient un chemin de fils en fils et de frères en frères. Pour savoir si un chemin est un mot valide, on rajoute un champs à chaque noeud

de notre arbre, une variable booléenne 'motValide' par exemple. Sur la figure ci-dessus, si notre arbre contient uniquement le mot 'yoyo', seulement le deuxième 'o' aura le champ 'motValide' à VRAI car 'y', 'yo' et 'yoy' ne sont pas des mots dans notre dictionnaire. Si on voulait ajouter le mot 'yoyos' il suffirait d'ajouter le fils 's' au deuxième 'o' et d'avoir les champs 'motValide' à VRAI pour le deuxième 'o' et le 's'.

5 Analyse

5.1 Types Abstraits de Données (TAD)

5.1.1 TAD Mot

Nom: Mot

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

Opérations:
- longueur: Mot → **Naturel**
- obtenirlemeCaractere: Mot × **NaturelNonNul** ⇸ **Caractere**
- fixerlemeCaractere: Mot × **NaturelNonNul** × **Caractere** ⇸ Mot
- remplacerLettre: Mot × **NaturelNonNul** × **Caractere** ⇸ Mot
- inverserDeuxLettresConsecutives: Mot × **NaturelNonNul** ⇸ Mot
- supprimerLettre: Mot × **NaturelNonNul** ⇸ Mot
- insererLettre: Mot × **Caractere** × **NaturelNonNul** ⇸ Mot
- obtenirMotEntreeStandard: Mot × **Entier** [1] → **Entier** [1] × Mot × **Booleen**
- fixerLaChaine: Mot × **Chaine de caracteres** → Mot
- obtenirLaChaine: Mot ⇸ **Chaine de caracteres**
- fixerLongueur: Mot × **Naturel** → Mot
- obtenirLongueur: Mot → **Naturel**
- sontEgaux: Mot × Mot → **Booleen**
- estUneLettre: **Caractere** → **Booleen**

Axiomes:
- supprimer(insererLettre(mot, c, i), i)=mot
- inverserDeuxLettresConsecutives(inverserDeuxLettresConsecutives(mot, pos), pos)=mot

Préconditions:
- obtenirlemeCaractere(mot, pos): $1 \leq$ pos $\leq$ obtenirLongueur(mot)
- fixerlemeCaractere(mot, pos, c): $1 \leq$ pos $\leq$ obtenirLongueur(mot)+1
- remplacerLettre(mot, pos, lettre): $1 \leq$ pos $\leq$ obtenirLongueur(mot)
- inverserDeuxLettresConsecutives(mot, pos): $1 \leq$ pos $\leq$ obtenirLongueur(mot)-1
- supprimerLettre(mot, pos): ($1 \leq$ pos $\leq$ obtenirLongueur(mot)) et (obtenirLongueur(mot) ≥ 2)
- insererLettre(mot, c, pos): $1 \leq$ pos $\leq$ obtenirLongueur(mot)+1
- obtenirLaChaine(mot): obtenirLongueur(mot)>=0

5.1.2 TAD Dictionnaire

Nom: Dictionnaire

Paramètre: Mot

Utilise: **Booleen**, **Naturel**

1. J'utilise un entier ici et non un naturel pour la position depuis le début de l'entrée standard car cette position doit commencer à 0 donc j'initialise cette position à -1 dans le main

Opérations: creerDico: $\rightarrow$ Dictionnaire
estVide: Dictionnaire $\rightarrow$ **Booleen**
insererMot: Dictionnaire $\times$ Mot $\rightarrow$ Dictionnaire
estPresent: Dictionnaire $\times$ Mot $\rightarrow$ **Booleen**
sauvegarderDico: **Chaine de caracteres** $\times$ Dictionnaire $\rightarrow$ Fichier Texte
chargerDico: **Chaine de caracteres** $\times$ Fichier Texte $\rightarrow$ Dictionnaire

Axiomes:
- insererMot(mot, insererMot(mot, dico))=insererMot(mot,dico)
- estPresent(mot, insererMot(mot, dico)

5.1.3 TAD CorrecteurOrthographique

Nom: CorrecteurOrthographique
Paramètre: Mot, Dictionnaire
Utilise: **Booleen**, Ensemble
Opérations: correcteurOrthographique: $\rightarrow$ CorrecteurOrthographique
proposerCorrection: CorrecteurOrthographique $\times \nrightarrow$ Ensemble<Mot>
estBienOrthographie: CorrecteurOrthographique $\times \nrightarrow$ **Booleen**
obtenirMot: CorrecteurOrthographique $\rightarrow$ Mot
obtenirDictionnaire: CorrecteurOrthographique $\rightarrow$ Dictionnaire
fixerMot: CorrecteurOrthographique $\times$ Mot $\rightarrow$ CorrecteurOrthographique
fixerDictionnaire: CorrecteurOrthographique $\times$ Dictionnaire $\rightarrow$ CorrecteurOrthographique
Préconditions: proposerCorrection(co): non estBienOrthographie(co)

5.1.4 TAD Arbre n-aire

Pour représenter notre TAD Dictionnaire nous avons choisis d'utiliser un arbre n-aire. Nous allons maintenant détailler ce TAD.

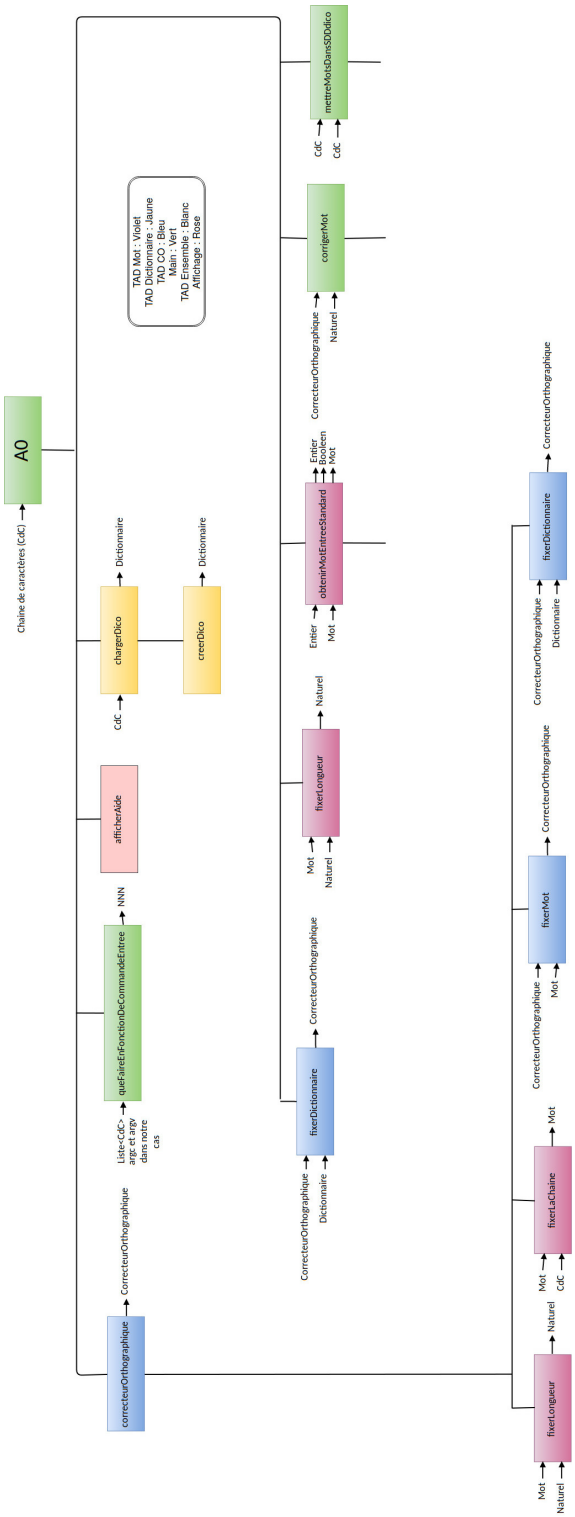
Nom: AbreN
Utilise: **Booleen**, **Chaine de caracteres**, **Caractere**, Ensemble
Opérations: creerArbreNonInit: $\rightarrow$ ArbreN
estVide: ArbreN $\rightarrow$ **Booleen**
obtenirBool: ArbreN $\nrightarrow$ **Booleen**
obtenirChar: ArbreN $\nrightarrow$ **Caractere**
fixerBool: ArbreN $\times$ **Booleen** $\nrightarrow$ ArbreN
fixerChar: ArbreN $\times$ **Caractere** $\nrightarrow$ ArbreN
fixerFils: ArbreN $\times$ ArbreN $\nrightarrow$ ArbreN
fixerFrere: ArbreN $\times$ ArbreN $\nrightarrow$ ArbreN

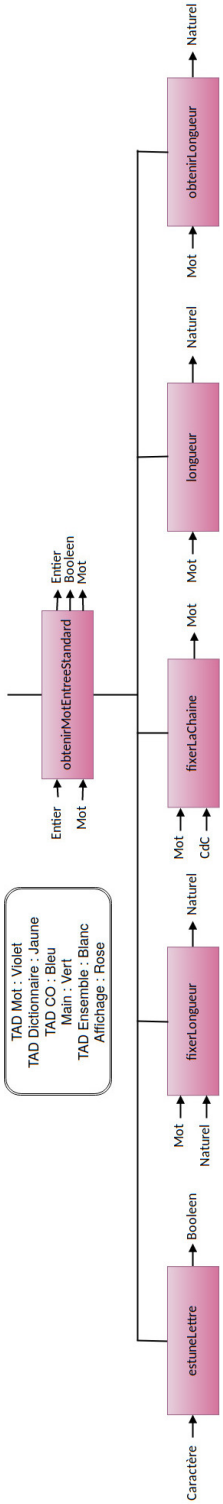
obtenirFils: ArbreN ⇸ ArbreN
obtenirFrere: ArbreN ⇸ ArbreN

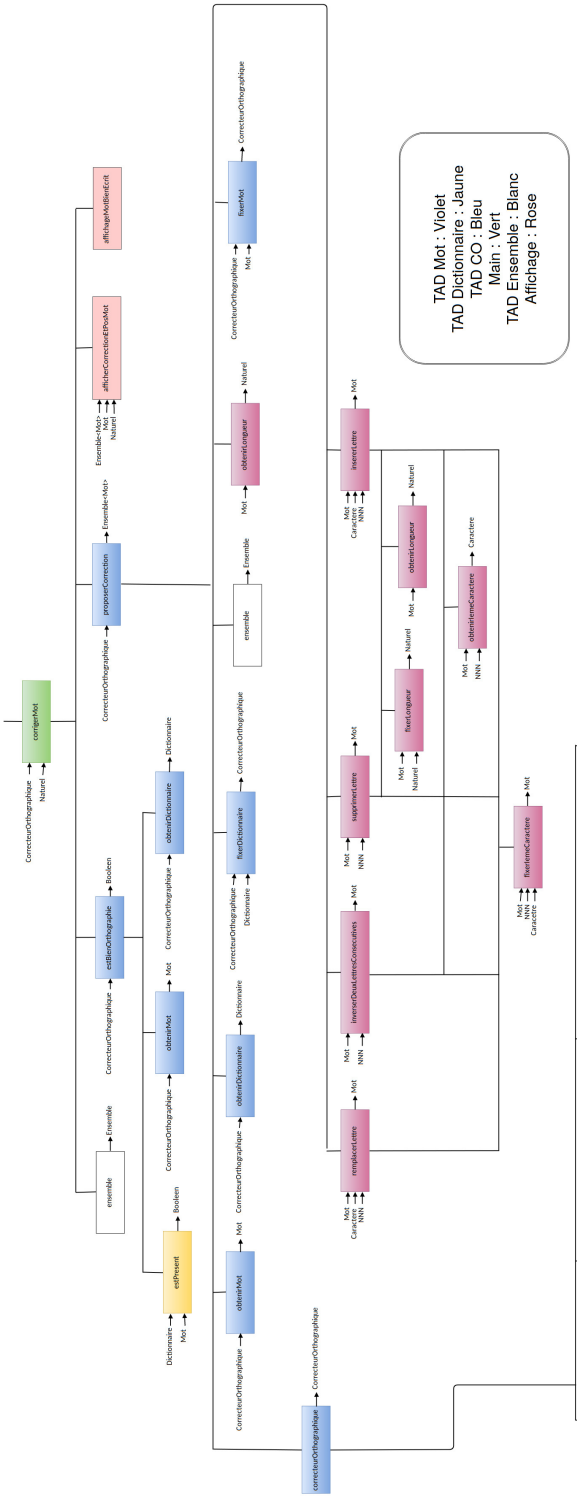
Préconditions: obtenirBool(arbre): non estVide(arbre)
obtenirChar(arbre): non estVide(arbre)
fixerBool(arbre, bool): non estVide(arbre)
fixerChar(arbre, car): non estVide(arbre)
fixerFils(arbre, fils): non estVide(arbre)
fixerFrere(arbre, frere): non estVide(arbre)
obtenirFils(arbre): non estVide(arbre)
obtenirFrere(arbre): non estVide(arbre)

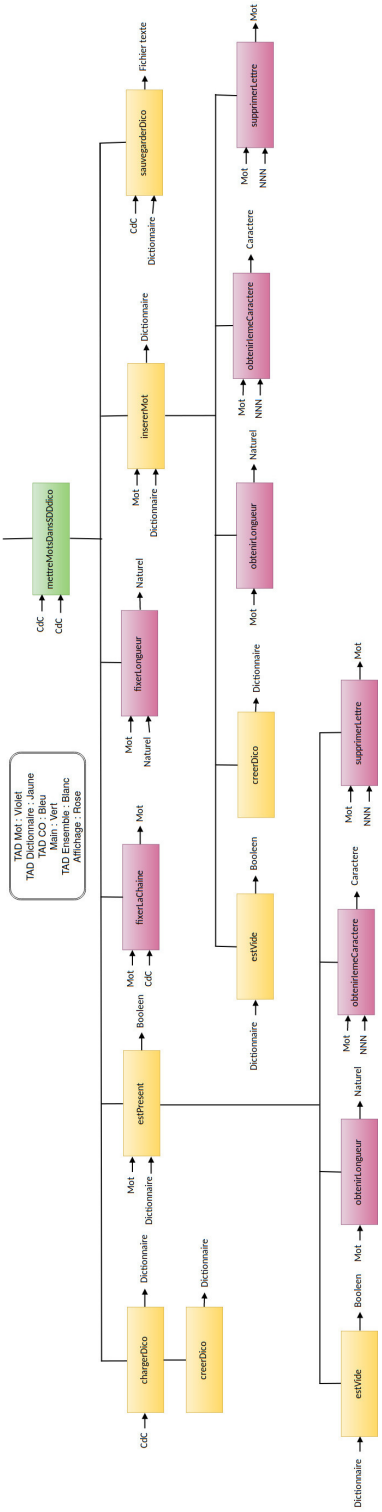
5.2 Analyse descendante

Notre analyse descendante se trouve sur les quatre pages suivantes. Du fait de sa taille, les figures ne sont pas très nettes. Nous vous invitons donc à regarder les images AnalyseDescendanteMAIN-90, AnalyseDescendanteMettreMotDansSDDdico-90, AnalyseDescendanteCorrigerMot-90, AnalyseDescendanteobtenirMotEntreeStandard-90 au format jpg dans le répertoire /rapport/images. Vous pouvez aussi cliquer *ici*, pour voir, en ligne, l'image de l'analyse descendante du programme principale, *ici* pour celle correspondant à 'mettreMotDansSDDdico', *ici* pour celle correspondant à 'corrigerMot', ou encore *ici* pour celle correspondant à 'obtenirMotEntreeStandard'.


A0
Chaine de caractères (CdC)
TAD Mot : Violet
TAD Dictionnaire : Jaune
TAD CC : Bleu
Main : Vert
TAD Ensemble : Blanc
Affichage : Rose
chargerDico
Dictionnaire
CdC
creerDico
Dictionnaire
afficherAide
queFaireEnFonctionDeCommandeEntree
NNN
Liste<CdC> argc et argv dans notre cas
correcteurOrthographique
CorrecteurOrthographique
mettreMotsDansSDDdico
CdC
CdC
corrigerMot
CorrecteurOrthographique
Naturel
obtenirMotEntreeStandard
Entier
Booleen
Mot
Entier
Mot
fixerLongueur
Naturel
Mot
Naturel
fixerDictionnaire
CorrecteurOrthographique
CorrecteurOrthographique
Dictionnaire
fixerDictionnaire
CorrecteurOrthographique
CorrecteurOrthographique
Dictionnaire
fixerMot
CorrecteurOrthographique
CorrecteurOrthographique
Mot
fixerLaChaine
Mot
Mot
CdC
fixerLongueur
Naturel
Mot
Naturel


TAD Mot : Violet
TAD Dictionnaire : Jaune
TAD CO : Bleu
Main : Vert
TAD Ensemble : Blanc
Affichage : Rose
obtenirMotEntreeStandard
Entier
Mot
Entier
Booleen
Mot
fixerLaChaine
Mot
CdC
Mot
longueur
Mot
Naturel
obtenirLongueur
Mot
Naturel
fixerLongueur
Mot
Naturel
Naturel
estuneLettre
Caractère
Booleen


TAD Mot : Violet
TAD Dictionnaire : Jaune
TAD CO : Bleu
Main : Vert
TAD Ensemble : Blanc
Affichage : Rose


TAD Mot : Violet
TAD Dictionnaire : Jaune
TAD IO : Bleu
Main : Vert
TAD Ensemble : Blanc
Affichage : Rose
mettreMotsDansSDDdico
CdC
CdC
sauvegarderDico
Fichier texte
CdC
Dictionnaire
chargerDico
CdC
Dictionnaire
creerDico
Dictionnaire
insererMot
Dictionnaire
Mot
Dictionnaire
fixerLongueur
Naturel
Mot
Naturel
fixerLaChaine
Mot
Mot
CdC
estPresent
Booleen
Mot
Dictionnaire
supprimerLettre
Mot
Mot
NNN
obtenirIemeCaractere
Caractere
Mot
NNN
obtenirLongueur
Naturel
Mot
creerDico
Dictionnaire
estVide
Booleen
Dictionnaire
supprimerLettre
Mot
Mot
NNN
obtenirIemeCaractere
Caractere
Mot
NNN
obtenirLongueur
Naturel
Mot
estVide
Booleen
Dictionnaire

6 Conception préliminaire

6.1 Signatures liées au TAD Mot

Type Mot = **Structure**
laChaine : **Chaine de caracteres**
longueur : **Naturel**
finstructure

fonction M.longueur (mot : Mot) **: Naturel**
fonction M.obtenirIemeCaractere (mot : Mot, pos : **NaturelNonNul**) **: Caractere**
⌊**précondition(s)** $1 \leq pos \leq$ M.obtenirLongueur(mot)
procédure M.fixerIemeCaractere (**E/S** mot : Mot, **E** pos : **NaturelNonNul**, lettre :**Caractere**)
⌊**précondition(s)** $1 \leq pos \leq$ M.obtenirLongueur(mot)+1
fonction M.remplacerLettre (mot : Mot, pos : **NaturelNonNul**, lettre : **Caractere**) **:** Mot
⌊**précondition(s)** $1 \leq pos \leq$ M.obtenirLongueur(mot)
fonction M.inverserDeuxLettresConsecutives (mot : Mot, pos : **NaturelNonNul**) **:** Mot
⌊**précondition(s)** $1 \leq pos \leq$ M.obtenirLongueur(mot)-1
fonction M.supprimerLettre (mot : Mot, pos : **NaturelNonNul**) **:** Mot
⌊**précondition(s)** $(1 \leq pos \leq$ obtenirLongueur(mot)) et (obtenirLongueur(mot) $\geq 2)$
fonction M.insererLettre (mot : Mot, lettre : **Caractere**, pos : **NaturelNonNul**) **:** Mot
⌊**précondition(s)** $1 \leq pos \leq$ M.obtenirLongueur(mot)+1
procédure M.obtenirMotEntreeStandard (**E/S** posDepuisDebutFlux : **Entier**[2], mot : Mot, **S** arret : **Booleen**)
procédure M.fixerLaChaine (**E/S** mot : Mot, **E** chaine : **Chaine de caracteres**)
fonction M.obtenirLaChaine (mot : Mot) **: Chaine de caracteres**
⌊**précondition(s)** M.obtenirLongueur(mot)>=0
procédure M.fixerLongueur (**E/S** mot : Mot, **E** length : **Naturel**)
fonction M.obtenirLongueur (mot : Mot) **: Naturel**
fonction M.sontEgaux (mot1, mot2 : Mot) **: Booleen**
fonction M.estUneLettre (c : **Caractere**) **: Booleen**

6.2 Signatures liées au TAD Dictionnaire

Type Dictionnaire = ArbreN

fonction D.creerDico () **:** Dictionnaire

2. J'utilise un entier ici et non un naturel pour la position depuis le début de l'entrée standard car cette position doit commencer à 0 donc j'initialise cette position à -1 dans le main

fonction D.estVide (dico : Dictionnaire) **: Booleen**
procédure D.insererMot (**E/S** dico : Dictionnaire, **E** lemot : Mot)
fonction D.estPresent (dico : Dictionnaire, mot : Mot) **: Booleen**
procédure D.sauvegarderDico (**S** fichierSDDDico : Fichier Texte, **E** dico : D.Dictionnaire)
fonction D.chargerDico (nomFichier : **Chaine de caracteres**, fichierSDDDico : Fichier Texte) **:** Dictionnaire

⌊**précondition(s)** F.fichierExiste(nomFichier)

6.3 Signatures liées au TAD CorrecteurOrthographique

Type CorrecteurOrthographique = **Structure**
mot : Mot
dico : Dictionnaire
finstructure

fonction CO.correcteurOrthographique () **:** CorrecteurOrthographique
fonction CO.proposerCorrection (co : CorrecteurOrthographique) **:** Ensemble<Mot>

⌊**précondition(s)** non estBienOrthographie(co)

fonction CO.estBienOrthographie (co : CorrecteurOrthographique) **: Booleen**
fonction CO.obtenirMot (co : CorrecteurOrthographique) **:** Mot
fonction CO.obtenirDictionnaire (co : CorrecteurOrthographique) **:** Dictionnaire
procédure CO.fixerMot (**E/S** co : CorrecteurOrthographique, **E** mot : Mot)
procédure CO.fixerDictionnaire (**E/S** co : CorrecteurOrthographique, **E** dico : Dictionnaire)

6.4 Signatures liées au TAD Arbre n-aire

Type ArbreN = ^Noeud
Type Noeud = **Structure**
lettre : **Caractere**
motValide : **Booleen**
Fils : ArbreN
Frere : ArbreN
finstructure

fonction AbN.creerArbreNonInit () **:** ArbreN
fonction AbN.estVide (arbre : ArbreN) **: Booleen**
fonction AbN.obtenirBool (arbre : ArbreN) **: Booleen**

⌊**précondition(s)** non AbN.estVide(arbre)

fonction AbN.obtenirChar (arbre : ArbreN) **: Caractere**

⌊**précondition(s)** non AbN.estVide(arbre)

procédure AbN.fixerBool (**E/S** arbre : ArbreN **E** valide : **Booleen**)

⌊**précondition(s)** non AbN.estVide(arbre)

procédure AbN.fixerChar (**E/S** arbre : ArbreN **E** lalettre : **Caractere**)

⌊**précondition(s)** non AbN.estVide(arbre)

procédure AbN.fixerFils (**E/S** arbre : ArbreN **E** fils : ArbreN)

⌊**précondition(s)** non AbN.estVide(arbre)

procédure AbN.fixerFrere (**E/S** arbre : ArbreN **E** frere : ArbreN)

⌊**précondition(s)** non AbN.estVide(arbre)

fonction AbN.obtenirFils (arbre : ArbreN) **:** ArbreN

⌊**précondition(s)** non AbN.estVide(arbre)

fonction AbN.obtenirFrere (arbre : ArbreN) **:** ArbreN

⌊**précondition(s)** non AbN.estVide(arbre)

6.5 Signatures des procédures d'affichage

procédure A.afficherAide ()
procédure A.affichageMotBienEcrit ()
procédure A.afficherCorrectionEtPosMot (**E** lesCorrections : Ensemble<Mot>, leMot : Mot, pos : **Naturel**)

6.6 Signatures des procédures privées du main

fonction queFaireEnFonctionDeCommandeEntree[3] (argc : **Naturel**, argv : tableau de CdC) **:** **NaturelNonNul**
procédure mettreMotsDansSDDdico (**E** nomFichierSDDDico : **Chaine de caracteres**, nomFichierTexteMotsAInserer : **Chaine de caracteres**)
procédure corrigerMot (**E** co : CorrecteurOrthographique, posMotFaux : **Naturel**)

6.7 Signatures liées au TAD Fichier Texte donné dans le sujet

fonction fichierTexte (chaine : **Chaine de caracteres**) **:** FichierTexte
procédure ouvrir (**E/S** f : FichierTexte, **E** Mode)

⌊**précondition(s)** non estOuvert(f)

procédure fermer (**E/S** f : FichierTexte)

⌊**précondition(s)** estOuvert(f)

fonction estOuvert (f : FichierTexte) **:** Booleen
fonction mode (f : FichierTexte) **:** Mode
fonction finFichier (f : FichierTexte) **:** Booleen

3. Nous ne mettons pas cette fonction en pseudocode dans la conception détaillée car elle est spécifique au langage utilisé (argc et agrv en C)

⌊**précondition(s)** mode(f)=lecture

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

⌊**précondition(s)** estOuvert(f) et mode(f)=ecriture

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

⌊**précondition(s)** estOuvert(f) et mode(f)=lecture et non finFichier(f)

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

⌊**précondition(s)** estOuvert(f) et mode(f)=ecriture

procédure lireCaractere (**E/S** f : FichierTexte, **S** c : **Caractere**)

⌊**précondition(s)** estOuvert(f) et mode(f)=lecture et non finFichier(f)

7 Conception détaillée

7.1 Fonctions et procédures liées au TAD Mot

7.1.1 Partie Privée

Constante LETTRES_SPECIALES_AUTORISEES = "ÀÂÄÆÇÈÉÊËÌÎÏÒÓÔÖÙÚÛÜàâäæçèéêëìîïòôöùûü"

Constante LONGUEUR_LETTRES_SPECIALES_AUTORISEES = 38

7.1.2 Partie Publique

Type Mot = **Structure**
  laChaine : **Chaine de caracteres**
  longueur : **Naturel**
finstructure

fonction M.longueur (mot : Mot) **: Naturel**
debut
  retourner longueur(mot.laChaine)
fin

fonction M.obtenirLongueur (mot : Mot) **: Naturel**
debut
  retourner mot.longueur
fin

Nous pouvons remarquer ici que les deux dernières fonctions semblent réaliser la même action. Or dans le premier cas ("M.longueur") la fonction est une fonction d'encapsulation qui utilise la fonction "strlen" de la bibliothèque C "string.h". Nous l'utilisons une fois dans le code pour fixer la longueur du mot la première fois. Ensuite nous ne nous servons plus que de l'accesseur "M.obtenirLongueur" qui renvoie le champ "longueur" de la structure Mot. En C, cet accesseur, nous permettra d'avoir accès à la longueur d'un mot en $\mathcal{O}(1)$.

fonction M.obtenirIemeCaractere (mot : Mot, pos : **NaturelNonNul**) **:** Mot
  ⌊**précondition(s)** $1 \leq$ pos $\leq$ M.obtenirLongueur(mot)
debut
  retourner mot[pos]
fin

procédure M.fixerIemeCaractere (**E/S** mot : Mot, **E** pos : **NaturelNonNul**, lettre : **Caractere**)
  ⌊**précondition(s)** $1 \leq$ pos $\leq$ M.obtenirLongueur(mot)+1

debut
mot[pos] ← lettre
fin

fonction M.remplacerLettre (mot : Mot, pos : **NaturelNonNul**, lettre : **Caractere**) **:** Mot
⌊**précondition(s)** 1 ≤ pos ≤ M.obtenirLongueur(mot)
debut
M.fixerIemeCaractere(mot, pos, lettre)
retourner mot
fin

fonction M.inverserDeuxLettresConsecutives (mot : Mot, pos : **NaturelNonNul**) **:** Mot
⌊**précondition(s)** 1 ≤ pos ≤ M.obtenirLongueur(mot)-1
Déclaration temp : **Caractere**
debut
temp ← M.obtenirIemeCaractere(mot,pos)
M.fixerIemeCaractere(mot, pos, M.obtenirIemeCaractere(mot, pos+1))
M.fixerIemeCaractere(mot, pos+1, temp)
retourner mot
fin

fonction M.supprimerLettre (mot : Mot, pos : **NaturelNonNul**) **:** Mot
⌊**précondition(s)** 1 ≤ pos ≤ M.obtenirLongueur(mot)
Déclaration i : **NaturelNonNul**
debut
pour i ←pos **à** M.obtenirLongueur(mot) **faire**
M.fixerIemeCaractere (mot, i, M.obtenirIemeCaractere(mot,i+1))
finpour
M.fixerLongueur(mot, M.obtenirLongueur(mot)-1)
M.fixerIemeCaractere(mot, M.obtenirLongueur(mot), '\0')
retourner mot
fin

fonction M.insererLettre (mot : Mot, lettre : **Caractere**, pos : **NaturelNonNul**) **:** Mot
⌊**précondition(s)** 1 ≤ pos ≤ M.obtenirLongueur(mot)+1
Déclaration i : **NaturelNonNul**
debut
pour i ←longueur(mot) **à** pos **pas de** -1 **faire**

```
        M.fixerIemeCaractere(mot, i+1, M.obtenirIemeCaractere(mot, i))
    finpour
    M.fixerIemeCaractere(mot, pos, lettre)
    M.fixerLongueur(mot, M.obtenirLongueur(mot)+1)
    M.fixerIemeCaractere(mot, M.obtenirLongueur(mot), '\0')
    retourner mot
fin

fonction M.estUneLettre (c : Caractere) : Booleen
    Déclaration  i : NaturelNonNul
                 estPresent : Booleen
debut
    i ← 1
    estPresent ← FAUX
    tant que (i<=LONGUEUR_LETTRES_SPECIALES_AUTORISEES) et (estPresent=FAUX)
    faire
        si (LETTRES_SPECIALES_AUTORISEES[i]=c) alors
            estPresent ← VRAI
        finsi
        i ← i+1
    fintantque
    retourner (estPresent ou ((c>='a') et (c<='z')) ou ((c>='A') et (c<='Z')) ou (c='-'))
fin

procédure M.obtenirMotEntreeStandard [4] (E/S posDepuisDebutFlux : Entier [5], mot : Mot, S arret :
Booleen)
    Déclaration  c : caractere
debut
    posDepuisDebutFlux ← posDepuisDebutFlux+naturelEnEntier( M.obtenirLongueur(mot))+1
    mot ← ""
    c ← obtenirCaractereSuivantEntreeStandard()
    tant que (non M.estUneLettre(c) et non estLaFinEntreeStandard(c)) faire
        c ← obtenirCaractereSuivantEntreeStandard()
        posDepuisDebutFlux ← posDepuisDebutFlux+1 ;
    fintantque
    tant que M.estUneLettre(c) et non estLaFinEntreeStandard(c) faire
```

4. Vous remarquerez que nous n'avons pas écrit de test unitaire pour cette procédure car nous ne savions pas comment faire. En effet, il aurait fallu exécuter le programme avec des paramètres au sein même du test unitaire. En revanche, nous avons testé cette procédure "à la main" dans de nombreux cas et elle passe nos tests

5. J'utilise un entier ici et non un naturel pour la position depuis le début de l'entrée standard car cette position doit commencer à 0 donc j'initialise cette position à -1 dans le main

```
        concatener(mot, majusculeEnMinuscule(caractereEnChaine(c)))
        c ← obtenirCaractereSuivantEntreeStandard()
    fintantque
    fixerLongueur(mot, longueur(mot))
    si estLaFinEntreeStandard(c) alors
        arret ← VRAI
    finsi
fin

procédure M.fixerLaChaine (E/S mot : Mot, E chaine : Chaine de caracteres)
debut
    mot.laChaine ← chaine
fin

fonction M.obtenirLaChaine (mot : Mot) : Chaine de caracteres
    ⌊précondition(s)   M.obtenirLongueur(mot)>=0
debut
    retourner mot.laChaine
fin

procédure M.fixerLongueur (E/S mot : Mot, E length : Naturel)
debut
    mot.longueur ← length
fin

fonction M.sontEgaux (mot1, mot2 : Mot) : Booleen
debut
    retourner M.obtenirLaChaine(mot1)=M.obtenirLaChaine(mot2)
fin
```

7.2 Fonctions et procédures liées au TAD Dictionnaire

7.2.1 Partie Privée

```
Constante CaratereMotValide = '*'
Constante CaratereMotNonValide = ','
Constante CaratereFilsNonVide = '/'
Constante CaratereFilsVide = '.'
Constante CaratereFrereNonVide = ' :'
Constante CaratereFrereVide = ';'

procédure serialiserParcoursRGDdico (E dico : D.Dictionnaire, E/S fichierSDDDico : Fichier Texte)
```

```
debut
    si non D.estVide(dico) alors
        ecrireCaractere(fichierSDDDico, AbN.obtenirChar(dico))
        si AbN.obtenirBool(dico)=VRAI alors
            ecrireCaractere(fichierSDDDico, CaratereMotValide)
        sinon
            ecrireCaractere(fichierSDDDico, CaratereMotNonValide)
        finsi
        si AbN.estVide(AbN.obtenirFils(dico)) alors
            ecrireCaractere(fichierSDDDico, CaratereFilsVide
        sinon
            ecrireCaractere(fichierSDDDico, CaratereFilsNonVide)
            serialiserParcoursRGDdico(AbN.obtenirFils(dico), fichierSDDDico)
        finsi
        si AbN.estVide(AbN.obtenirFrere(dico)) alors
            ecrireCaractere(fichierSDDDico, CaratereFrereVide
        sinon
            ecrireCaractere(fichierSDDDico, CaratereFrereNonVide)
            serialiserParcoursRGDdico(AbN.obtenirFrere(dico), fichierSDDDico)
        finsi
    finsi
fin
procédure deserialiser (E/S dico : Dictionnaire, E fichierSDDDico : Fichier Texte)
    Déclaration   temp : Dictionnaire
                  c : Caractere
debut
    lireCaractere(fichierSDDDico, c)
    si non finFichier(fichierSDDDico alors
        AbN.fixerChar(dico, c)
        lireCaractere(fichierSDDDico, c)
        si c=CaratereMotValide alors
            AbN.fixerBool(dico, VRAI)
        sinon
            AbN.fixerBool(dico, FAUX)
        finsi
        lireCaractere(fichierSDDDico, c)
        si c=CaratereFilsNonVide alors
            temp ← D.creerDico()
            deserialiser(temp, fichierSDDDico)
            dico.Fils ← temp
        finsi
```

```
        lireCaractere(fichierSDDDico, c)
        si c=CaratereFrereNonVide alors
            temp ← D.creerDico()
            deserialiser(temp, fichierSDDDico)
            dico.Frere ← temp
        finsi
    finsi
fin
```

7.2.2 Partie publique

```
fonction D.creerDico () : Dictionnaire
    Déclaration   dico : Dictionnaire
debut
    dico ← AbN.creerArbreNonInit()
    retourner dico
fin

fonction D.estVide (dico : Dictionnaire) : Booleen
debut
    retourner AbN.estVide(dico)
fin

procédure D.insererMot (E mot : Mot, E/S dico : Dictionnaire)
    Déclaration   temp, newNoeud, newNoeudTemp : Dictionnaire
                  tempMot : Mot
debut
    si D.estVide(dico) alors
        dico ← D.creerDico()
        AbN.fixerChar(dico, M.obtenirIemeCaractere(tempMot,0))
        AbN.fixerBool(dico, M.obtenirLongueur(tempMot)=1)
        si M.obtenirLongueur(tempMot)>1 alors
            temp ← AbN.obtenirFils(dico)
            D.insererMot(temp,M.supprimerLettre(tempMot,0))
            AbN.fixerFils(dico, temp)
        finsi
    sinon
        si AbN.obtenirChar(dico) = M.obtenirIemeCaractere(tempMot,0) alors
            si M.obtenirLongueur(tempMot) = 1 alors
                AbN.fixerBool (dico, VRAI)
            sinon
```

```
            temp ← AbN.obtenirFils(dico)
            D.insererMot(temp,M.supprimerLettre(tempMot,0))
            AbN.fixerFils(dico, temp)
        finsi
    sinon
        si AbN.obtenirChar(dico) < M.obtenirIemeCaractere(tempMot,0) alors
            temp ← AbN.obtenirFrere(dico)
            D.insererMot(temp,tempMot)
            AbN.fixerFrere(dico, temp)
        sinon
            temp ← dico
            newNoeud ← D.creerDico()
            AbN.fixerChar (newNoeud, M.obtenirIemeCaractere(tempMot,0))
            AbN.fixerBool (newNoeud, M.obtenirLongueur(tempMot)=1)
            si M.obtenirLongueur(tempMot)=1 alors
                AbN.fixerFrere(newNoeud, temp)
                AbN.fixerFils(newNoeud, newNoeudTemp)
                dico ← newNoeud
            sinon
                newNoeudTemp ← AbN.obtenirFils(newNoeud)
                D.insererMot(newNoeudTemp, M.supprimerLettre(tempMot,0))
                AbN.fixerFrere(newNoeud, temp)
                AbN.fixerFils(newNoeud, newNoeudTemp)
                dico ← newNoeud
            finsi
        finsi
    finsi
finsi
fin

fonction D.estPresent (mot : Mot, dico : Dictionnaire) : Booleen
    Déclaration  temp : Mot
debut
    si D.estVide(dico) alors
        retourner FAUX
    sinon
        si ((M.obtenirLongueur(temp)=1) et (AbN.obtenirChar(dico) = M.obtenirIemeCaractere(temp,0))
        et (AbN.obtenirBool(dico)=VRAI)) alors
            retourner VRAI
        sinon
            si ((AbN.obtenirChar(dico) = M.obtenirIemeCaractere(temp,0) et (M.obtenirLongueur(temp)>1)))
```

```
            alors
                retourner D.estPresent(AbN.obtenirFils(dico), M.supprimerLettre(temp,0))
            sinon
                si (AbN.obtenirChar(dico) < M.obtenirIemeCaractere(temp,0)) alors
                    retourner D.estPresent(AbN.obtenirFrere(dico),temp)
                sinon
                    retourner FAUX
                finsi
            finsi
        finsi
    finsi
fin

procédure D.sauvegarderDico (S fichierSDDDico : Fichier Texte, E dico : D.Dictionnaire)
debut
    ouvrir(fichierSDDDico, ecriture)
    serialiserParcoursRGDdico(dico, fichierSDDDico)
    fermer(fichierSDDDico)
fin
fonction D.chargerDico (nomFichier : Chaine de caracteres, fichierSDDDico : Fichier Texte) :
Dictionnaire
    ⌊précondition(s)  F.fichierExiste(nomFichier)
    Déclaration  dico : Dictionnaire
debut
    ouvrir(fichierSDDDico, lecture)
    dico ← D.creerDico()
    deserialiser(dico, fichierSDDDico)
    fermer(fichierSDDDico)
    retourner dico
fin
```

7.3 Fonctions et procédures liées au TAD CorrecteurOrthographique

7.3.1 Partie privée

```
Constante LETTRES_SPECIALES_AUTORISEES = "ÀÂÄÆÇÈÉÊËÌÎÏÒÓÔÖÙÚÛÜàâäæçèéêëìîïòôöùûü"

Constante LONGUEUR_LETTRES_SPECIALES_AUTORISEES = 38

procédure ajouterSiCorrecte (E/S corrections : Ensemble<Mot>, E co : CorrecteurOrthographique)
debut
    si Co.estBienOrthographie(co) alors
```

```
        E.ajouter(corrections, obtenirMot(co))
    finsi
fin
```

7.3.2 Partie Publique

```
fonction CO.proposerCorrection (co : CorrecteurOrthographique) : Ensemble<Mot>
    ⌊précondition(s)   non CO.estBienOrthographie(co)
    Déclaration   i,k : NaturelNonNul
                  j : Caractere
                  temp, mot : Mot
                  dico : Dictionnaire
                  coTest : CorrecteurOrthographique
                  corrections : Ensemble
debut
    coTest ← CO.correcteurOrthographique()
    mot ← CO.obtenirMot(co)
    dico ← CO.obtenirDictionnaire(co)
    corrections ← E.ensemble()
    si M.obtenirLongueur(mot)>=2 alors
        pour i ←1 à M.obtenirLongueur(mot) faire
            temp ← M.inverserDeuxLettreConsecutives(mot, i)
            CO.fixerMot(coTest,temp)
            ajouterSiCorrecte(corrections,coTest)
        finpour
        pour i ←1 à M.obtenirLongueur(mot)+1 faire
            temp ← M.supprimerLettre(mot, i)
            CO.fixerMot(coTest,temp)
            ajouterSiCorrecte(corrections,coTest)
        finpour
    finsi
    pour j ←'a' à 'z' faire
        pour i ←1 à M.obtenirLongueur(mot) faire
            temp ← M.remplacerLettre(mot, i, j)
            CO.fixerMot(coTest,temp)
            ajouterSiCorrecte(corrections,coTest)
        finpour
    finpour
    tant que k <= LONGUEUR_LETTRES_SPECIALES_AUTORISEES faire
        pour i ←1 à M.obtenirLongueur(mot)+1 faire
```

```
            temp ← M.remplacerLettre(mot, i, LETTRES_SPECIALES_AUTORISEES[k])
            CO.fixerMot(coTest,temp)
            ajouterSiCorrecte(corrections,coTest)
        finpour
        k = k+1
    fintantque
    pour j ←'a' à 'z' faire
        pour i ←1 à M.obtenirLongueur(mot)+1 faire
            temp ← M.insererLettre(mot, j, i)
            CO.fixerMot(coTest,temp)
            ajouterSiCorrecte(corrections,coTest)
        finpour
    finpour
    tant que k <= LONGUEUR_LETTRES_SPECIALES_AUTORISEES faire
        pour i ←1 à M.obtenirLongueur(mot)+1 faire
            temp ← M.insererLettre(mot, LETTRES_SPECIALES_AUTORISEES[k], i)
            CO.fixerMot(coTest,temp)
            ajouterSiCorrecte(corrections,coTest)
        finpour
        k = k+1
    fintantque
    retourner corrections
fin

fonction CO.estBienOrthographie (co : CorrecteurOrthographique) : Booleen
debut
    retourner CO.estPresent(obtenirMot(co), CO.obtenirDictionnaire(co))
fin

fonction CO.obtenirMot (co : CorrecteurOrthographique) : Mot
debut
    retourner co.mot
fin

fonction CO.obtenirDictionnaire (co : CorrecteurOrthographique) : Dictionnaire
debut
    retourner co.dico
fin

fonction CO.correcteurOrthographique () : CorrecteurOrthographique
    Déclaration   co : CorrecteurOrthographique
```

```
                mot : Mot
                dico : Dictionnaire
debut
    M.fixerLaChaine(mot, "")
    M.fixerLongueur(mot, 0)
    CO.fixerMot(co, mot)
    dico ← NULL
    CO.fixerDictionnaire(co, dico)
    retourner co
fin
procédure CO.fixerMot (E/S co : CorrecteurOrthographique, E mot : Mot)
debut
    co.mot ← mot
fin
procédure CO.fixerDictionnaire (E/S co : CorrecteurOrthographique, E dico : Dictionnaire)
debut
    co.dico ← dico
fin
```

7.4 Fonctions et procédures liées au TAD ArbreN

```
fonction AbN.creerArbreNonInit () : ArbreN
    Déclaration   temp :ArbreN
debut
    temp.Fils ← NULL
    temp.Frere ← NULL
    temp.lettre ← "caractereVide"
    temp.motValide ← 0
    retourner temp
fin

fonction AbN.estVide (arbre :ArbreN) : Booleen
debut
    retourner (arbre=NULL) ou (arbre.lettre="caractereNul")
fin

fonction AbN.obtenirBool (arbre : ArbreN) : Booleen
    ⌊précondition(s)   non AbN.estVide(arbre)
debut
    retourner (arbre.motValide)
```

fin

fonction AbN.obtenirChar (arbre : ArbreN) **: Caractere**
⌊**précondition(s)** non AbN.estVide(arbre)
debut
retourner (arbre.lettre)
fin

procédure AbN.fixerBool (**E/S** arbre : ArbreN **E** valide : **Booleen**)
⌊**précondition(s)** non AbN.estVide(arbre)
debut
arbre.motValide ← valide
fin
procédure AbN.fixerChar (**E/S** arbre : ArbreN **E** lalettre : **Booleen**)
⌊**précondition(s)** non AbN.estVide(arbre)
debut
arbre.lettre ← lalettre
fin
procédure AbN.fixerFils (**E/S** arbre : ArbreN **E** fils : ArbreN)
⌊**précondition(s)** non AbN.estVide(arbre)
debut
arbre.Fils ← fils
fin
procédure AbN.fixerFrere (**E/S** arbre : ArbreN **E** frere : ArbreN)
⌊**précondition(s)** non AbN.estVide(arbre)
debut
arbre.Frere ← frere
fin
fonction AbN.obtenirFils (arbre : ArbreN) **:** ArbreN
⌊**précondition(s)** non AbN.estVide(arbre)
debut
retourner arbre.Fils
fin

fonction AbN.obtenirFrere (arbre : ArbreN) **:** ArbreN
⌊**précondition(s)** non AbN.estVide(arbre)
debut
retourner arbre.Frere
fin

7.5 Procédures d'affichage

7.5.1 Partie privée

Constante NOM_FICHIER_AIDE = "./fichierAide.txt"

7.5.2 Partie publique

procédure A.afficherCorrectionEtPosMot (**E** lesCorrections : Ensemble<Mot>, leMot : Mot, pos : **Naturel**)

```
   Déclaration  i, nbCorrections : Naturel
                mot : Mot
                lesMotsCorrects : ListeChaineeMot
debut
   lesMotsCorrects ← E.obtenirLesElements(lesCorrections)
   nbCorrections ← E.cardinalite(lesCorrections)
   ecrire("&" M.obtenirLaChaine(leMot), naturelEnChaine(nbCorrections), naturelEnChaine(pos))
   pour i ←0 à i<nbCorrections faire
      mot ← LCM.obtenirElement(lesMotsCorrects)
      ecrire(M.obtenirLaChaine(mot))
      lesMotsCorrects ← LCM.obtenirListeSuivante(lesMotsCorrects)
   finpour
fin
```

procédure A.affichageMotBienEcrit ()

```
debut
   ecrire("*")
fin
```

procédure A.afficherAide ()

```
   Déclaration  fichierAide : Fichier Texte
                chaine : Chaine de caracteres
debut
   si F.fichierExiste(NOM_FICHIER_AIDE) alors
      ouvrir(fichierAide, lecture)
      lireChaine(fichierAide, chaine)
      tant que non finFichier(fichierAide) faire
         ecrire(chaine)
      fintantque
      fermer(fichierAide)
   sinon
```

```
        ecrire("Le fichier d'aide est introuvable")
    finsi
fin
```

7.6 Procédures privées du main

procédure corrigerMot **(E** co : CorrecteurOrthographique, posMotFaux : **Naturel)**

```
    Déclaration   correctionsPossibles : Ensemble
                  temp : ListeChaineeMot
debut
    correctionsPossibles ← ensemble()
    si non CO.estBienOrthographie(co) alors
        correctionsPossibles ← CO.proposerCorrections(co)
        A.afficherCorrectionEtPosMot(correctionsPossibles, CO.obtenirMot(co), posMotFaux)
    sinon
        A.affichageMotBienEcrit()
    finsi
    temp ← E.obtenirLesElements(correctionsPossibles)
    liberer(temp)
fin
```

procédure mettreMotsDansSDDdico **((E** nomFichierSDDDico : **Chaine de caracteres**, fichierTexteMotsAInserer : Fichier Texte**)**

```
    Déclaration   mot : Mot
                  SDDDico, fils : Dictionnaire
                  temp : Chaine de caracteres
                  longueurMot : Naturel
debut
    SDDDico ← D.creerDico()
    si F.fichierExiste(nomFichierSDDDico) alors
        SDDDico ← D.chargerDico(nomFichierSDDDico)
        AbN.fixerFils(SDDDico, fils)
    sinon
        fils ← AbN.obtenirFils(SDDDico)
    finsi
    ouvrir(fichierTexteMotsAInserer, lecture)
    tant que non finFichier(fichierTexteMotsAInserer) faire
        lireChaine(fichierTexteMotsAInserer, temp)
        longueurMot ← longueur(temp)
        M.fixerLongueur(mot, longueurMot)
        M.fixerLaChaine(mot, temp)
```

```
        si non D.estPresent(fils, mot) alors
            D.insererMot(fils, mot)
        finsi
    fintantque
    fermer(fichierTexteMotsAInserer)
    AbN.fixerFils(SDDDico, fils)
    D.sauvegarderDico(nomFichierSDDDico, fils)
    liberer(fils)
    liberer(SDDDico)
fin
```

8 Code C

8.1 mot.h

```
1
2  #ifndef __mot__
3  #define __mot__
4
5  #define TAILLE_MAX_MOT 40
6
7  typedef struct M_Mot {
8    char laChaine[TAILLE_MAX_MOT];
9    unsigned int longueur;
10 } M_Mot;
11
12 unsigned int M_longueur (M_Mot mot);
13
14 char M_obtenirIemeCaractere (M_Mot mot, unsigned int pos);
15
16 void M_fixerIemeCaractere (M_Mot *mot, unsigned int pos, char lettre);
17
18 M_Mot M_remplacerLettre (M_Mot mot, unsigned int pos, char lettre);
19
20 M_Mot M_inverserDeuxLettresConsecutives (M_Mot mot, unsigned int pos);
21
22 M_Mot M_supprimerLettre (M_Mot mot, unsigned int pos);
23
24 M_Mot M_insererLettre (M_Mot mot, char lettre, unsigned int pos);
25
26
27 void M_obtenirMotEntreeStandard (int *posDepuisDebutFlux, M_Mot *mot, int *arret);
28
29 void M_fixerLaChaine (M_Mot *mot, char *chaine);
30
31 char* M_obtenirLaChaine(M_Mot mot);
32
33 void M_fixerLongueur(M_Mot *mot, unsigned int length);
34
35 unsigned int M_obtenirLongueur(M_Mot mot);
36
37 int M_sontEgaux (M_Mot mot1, M_Mot mot2);
38
```

```
39 int M_estUneLettre (char c);
40 #endif
```

8.2 mot.c

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include <stdlib.h>
 4 #include <ctype.h>
 5 #include <assert.h>
 6 #include "mot.h"
 7
 8 #define NDEBUG
 9
10 #define TRUE 1
11 #define FALSE 0
12 #define LETTRES_SPECIALES_AUTORISEES "ÀÂÄÆÇÈÉÊËÌÎÏÒÓÔÖÙÚÛÜàâäæçèéêëìîïòôöùûü"
13 #define LONGUEUR_LETTRES_SPECIALES_AUTORISEES 38
14
15 int M_estUneLettre (char c) {
16   unsigned int i;
17   int estPresent;
18
19   i=0;
20   estPresent=FALSE;
21
22   while ((i<=LONGUEUR_LETTRES_SPECIALES_AUTORISEES-1) &&
23          (estPresent==FALSE)){
24     if (LETTRES_SPECIALES_AUTORISEES[i]==c) {
25       estPresent=TRUE;
26     }
27     i=i+1;
28   }
29   return (estPresent || ((c>='a') && (c<='z')) || ((c>='A') && (c<='Z'))
30           || (c=='-'));
31 }
32
33
34 void M_obtenirMotEntreeStandard (int *posDepuisDebutFlux, M_Mot *mot,
35                                  int *arret) {
36   int c;
37   char str[2];
```

```
38    char temp[TAILLE_MAX_MOT]="";
39    unsigned int i=0;
40
41    str[1]='\0';
42    *posDepuisDebutFlux=*posDepuisDebutFlux+(int)M_obtenirLongueur(*mot)+1;
43    c=getchar();
44
45    while ((!(M_estUneLettre((char)c))) && (c != EOF)) {
46      c=getchar();
47      *posDepuisDebutFlux=*posDepuisDebutFlux+1;
48    }
49
50    while (M_estUneLettre((char)c) && (c != EOF) && i<TAILLE_MAX_MOT-1) {
51      str[0]=(char)(tolower(c));
52      strcat(temp, str);
53      c=getchar();
54      i=i+1;
55    }
56
57    M_fixerLaChaine(mot, temp);
58    M_fixerLongueur(mot, M_longueur(*mot));
59
60    if (c == EOF) {
61      *arret=TRUE;
62    }
63  }
64
65  void M_fixerLongueur(M_Mot *mot, unsigned int length) {
66    mot->longueur=length;
67  }
68
69  unsigned int M_obtenirLongueur(M_Mot mot) {
70    return mot.longueur;
71  }
72
73  char* M_obtenirLaChaine(M_Mot mot) {
74    assert(M_obtenirLongueur(mot)>=0);
75
76    char *temp=(char*)malloc((M_obtenirLongueur(mot)+1)*sizeof(char));
77    strcpy(temp, mot.laChaine);
78    return temp;
79  }
```

```
80
81 void M_fixerLaChaine (M_Mot *mot, char *chaine) {
82   strcpy(mot->laChaine, chaine);
83 }
84
85 unsigned int M_longueur(M_Mot mot) {
86
87   return strlen(mot.laChaine);
88 }
89
90 char M_obtenirIemeCaractere (M_Mot mot, unsigned int pos) {
91   assert((0 <= pos) && (pos <= M_obtenirLongueur(mot)-1));
92
93   return mot.laChaine[pos];
94 }
95
96 void M_fixerIemeCaractere (M_Mot *mot, unsigned int pos, char lettre) {
97   assert((0 <= pos) && (pos <= M_obtenirLongueur(*mot)));
98
99   mot->laChaine[pos]=lettre;
100 }
101
102 M_Mot M_remplacerLettre (M_Mot mot, unsigned int pos, char lettre) {
103   assert((0 <= pos) && (pos <= M_obtenirLongueur(mot)-1));
104
105   M_Mot temp=mot;
106
107   M_fixerIemeCaractere (&temp, pos, lettre);
108   return temp;
109 }
110
111 M_Mot M_inverserDeuxLettresConsecutives (M_Mot mot, unsigned int pos) {
112   assert((0 <= pos) && (pos <= M_obtenirLongueur(mot)-2));
113
114   char tempCar;
115   M_Mot temp=mot;
116   tempCar=M_obtenirIemeCaractere (temp,pos);
117
118   M_fixerIemeCaractere (&temp, pos, M_obtenirIemeCaractere (temp, pos+1));
119   M_fixerIemeCaractere (&temp, pos+1, tempCar);
120
121   return temp;
```

```
122 }
123
124 M_Mot M_supprimerLettre (M_Mot mot, unsigned int pos){
125   assert(((0 <= pos) && (pos <= M_obtenirLongueur(mot)-1)) &&
126          (M_obtenirLongueur(mot)>=2));
127
128   int i;
129   M_Mot temp=mot;
130
131   for (i=pos; i<=M_obtenirLongueur(temp)-2; i++) {
132     M_fixerIemeCaractere (&temp, i, M_obtenirIemeCaractere(temp, i+1));
133   }
134
135   M_fixerLongueur(&temp, M_obtenirLongueur(temp)-1);
136   M_fixerIemeCaractere (&temp, M_obtenirLongueur(temp), '\0');
137   return temp;
138 }
139
140 M_Mot M_insererLettre (M_Mot mot, char lettre, unsigned int pos){
141   assert((0 <= pos) && (pos <= M_obtenirLongueur(mot)));
142
143   int i;
144   M_Mot temp=mot;
145
146   for (i=(int)M_obtenirLongueur(temp)-1; i>=(int)pos; i--) {
147
148     M_fixerIemeCaractere (&temp, i+1, M_obtenirIemeCaractere(temp, i));
149   }
150
151   M_fixerIemeCaractere (&temp, pos, lettre);
152   M_fixerLongueur(&temp, M_obtenirLongueur(temp)+1);
153   M_fixerIemeCaractere (&temp, M_obtenirLongueur(temp), '\0');
154   return temp;
155 }
156
157 int M_sontEgaux (M_Mot mot1, M_Mot mot2) {
158   return ((strcmp(M_obtenirLaChaine(mot1), M_obtenirLaChaine(mot2))==0) &&
159           (M_obtenirLongueur(mot1)==M_obtenirLongueur(mot2)));
160 }
```

8.3 dictionnaire.h

```

#ifndef __dictionnaire__
#define __dictionnaire__

#include "arbreN.h"


typedef AbN_ArbreN D_Dictionnaire;


D_Dictionnaire D_creerDico ();


int D_estVide (D_Dictionnaire dico);


int D_estPresent (D_Dictionnaire dico, M_Mot mot);


void D_insererMot(D_Dictionnaire* dico, M_Mot lemot);


void D_sauvegarderDico (char nomFichier[], D_Dictionnaire dico);


D_Dictionnaire D_chargerDico (char nomFichier[]);

#endif
```

8.4 dictionnaire.c

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "arbreN.h"
#include "mot.h"
#include "dictionnaire.h"
#include "existeFichier.h"

#define NDEBUG

```

```
11 #define CaratereMotValide '*'
12 #define CaratereMotNonValide ','
13 #define CaratereFilsNonVide '/'
14 #define CaratereFilsVide '.'
15 #define CaratereFrereNonVide ':'
16 #define CaratereFrereVide ';'
17
18 #define TRUE 1
19 #define FALSE 0
20 #define charnul '\0'
21
22
23 void serialiserParcoursRGDdico (D_Dictionnaire dico, FILE *fichierSDDDico) {
24
25   if (!(D_estVide(dico))) {
26     fputc((int)AbN_obtenirChar(dico), fichierSDDDico);
27
28     if (AbN_obtenirBool(dico)==TRUE) {
29       fputc((int)CaratereMotValide, fichierSDDDico);
30     }
31     else {
32       fputc((int)CaratereMotNonValide, fichierSDDDico);
33     }
34
35     if (AbN_estVide(AbN_obtenirFils(dico))) {
36       fputc((int)CaratereFilsVide, fichierSDDDico);
37     }
38     else {
39       fputc((int)CaratereFilsNonVide, fichierSDDDico);
40       serialiserParcoursRGDdico(AbN_obtenirFils(dico), fichierSDDDico);
41     }
42
43     if (AbN_estVide(AbN_obtenirFrere(dico))) {
44       fputc((int)CaratereFrereVide, fichierSDDDico);
45     }
46     else {
47       fputc((int)CaratereFrereNonVide, fichierSDDDico);
48       serialiserParcoursRGDdico(AbN_obtenirFrere(dico), fichierSDDDico);
49     }
50   }
51 }
52
```

```
53 void deserialiser (D_Dictionnaire *dico, FILE *fichierSDDDico) {
54   int c;
55   D_Dictionnaire temp;
56
57
58   if ((c=fgetc(fichierSDDDico)) != EOF) {
59     AbN_fixerChar(dico, (char)c);
60
61
62     c=fgetc(fichierSDDDico);
63
64     if (c==CaratereMotValide) {
65       AbN_fixerBool(dico, TRUE);
66     }
67     else {
68       AbN_fixerBool(dico, FALSE);
69     }
70
71
72     c=fgetc(fichierSDDDico);
73
74     if (c==CaratereFilsNonVide) {
75       temp = D_creerDico();
76       deserialiser(&temp, fichierSDDDico);
77       AbN_fixerFils(dico, temp);
78     }
79
80
81     c=fgetc(fichierSDDDico);
82
83     if (c==CaratereFrereNonVide) {
84       temp = D_creerDico();
85       deserialiser(&temp, fichierSDDDico);
86       AbN_fixerFrere(dico, temp);
87     }
88   }
89 }
90
91
92 D_Dictionnaire D_creerDico (){
93   D_Dictionnaire dico;
94   dico=AbN_creerArbreNonInit();
```

```
95    return dico;
96  }
97
98  int D_estVide (D_Dictionnaire dico){
99    return AbN_estVide(dico);
100 }
101
102 int D_estPresent (D_Dictionnaire dico, M_Mot mot) {
103   M_Mot temp=mot;
104   if (D_estVide(dico)) {
105     return FALSE;
106   }
107   else {
108     if ((M_obtenirLongueur(temp)==1) && (AbN_obtenirChar(dico) ==
109                                          M_obtenirIemeCaractere(temp,0)) &&
110         (AbN_obtenirBool(dico)==TRUE)) {
111       return TRUE;
112     }
113     else {
114       if ((AbN_obtenirChar(dico) == M_obtenirIemeCaractere(temp,0) &&
115            (M_obtenirLongueur(temp)>1))) {
116         return D_estPresent(AbN_obtenirFils(dico),M_supprimerLettre(temp,0));
117       }
118       else {
119         if (AbN_obtenirChar(dico) < M_obtenirIemeCaractere(temp,0)) {
120           return D_estPresent(AbN_obtenirFrere(dico),temp);
121         }
122         else {
123           return FALSE;
124         }
125       }
126     }
127   }
128 }
129
130 void D_insererMot (D_Dictionnaire *dico, M_Mot mot) {
131   D_Dictionnaire temp, newNoeud, newNoeudTemp;
132   M_Mot tempMot=mot;
133
134   if (D_estVide(*dico)) {
135     *dico = D_creerDico();
136     AbN_fixerChar (dico, M_obtenirIemeCaractere(tempMot,0));
```

```
137       AbN_fixerBool (dico, M_obtenirLongueur(tempMot)==1);
138
139       if (M_obtenirLongueur(tempMot)>1) {
140         temp = AbN_obtenirFils(*dico);
141         D_insererMot(&temp,M_supprimerLettre(tempMot,0));
142         AbN_fixerFils(dico, temp);
143       }
144     }
145     else {
146       if (AbN_obtenirChar(*dico) == M_obtenirIemeCaractere(tempMot,0)) {
147         if (M_obtenirLongueur(tempMot) == 1) {
148           AbN_fixerBool (dico, TRUE);
149         }
150         else {
151           temp = AbN_obtenirFils(*dico);
152           D_insererMot(&temp,M_supprimerLettre(tempMot,0));
153           AbN_fixerFils(dico, temp);
154         }
155       }
156       else {
157         if (AbN_obtenirChar(*dico) < M_obtenirIemeCaractere(tempMot,0)) {
158           temp = AbN_obtenirFrere(*dico);
159           D_insererMot(&temp,tempMot);
160           AbN_fixerFrere(dico, temp);
161         }
162         else {
163           temp = *dico;
164           newNoeud = D_creerDico();
165           AbN_fixerChar (&newNoeud, M_obtenirIemeCaractere(tempMot,0));
166           AbN_fixerBool (&newNoeud, M_obtenirLongueur(tempMot)==1);
167
168           if (M_obtenirLongueur(tempMot)==1) {
169             AbN_fixerFrere(&newNoeud, temp);
170             AbN_fixerFils(&newNoeud, newNoeudTemp);
171             *dico=newNoeud;
172           }
173           else {
174             newNoeudTemp = AbN_obtenirFils(newNoeud);
175             D_insererMot(&newNoeudTemp,M_supprimerLettre(tempMot,0));
176             AbN_fixerFrere(&newNoeud, temp);
177             AbN_fixerFils(&newNoeud, newNoeudTemp);
178             *dico=newNoeud;
```

```
179           }
180         }
181       }
182     }
183   }
184
185   void D_sauvegarderDico (char nomFichier[], D_Dictionnaire dico) {
186     FILE *fichierSDDDico;
187
188     fichierSDDDico=fopen(nomFichier, "w");
189     serialiserParcoursRGDdico(dico, fichierSDDDico);
190     fclose(fichierSDDDico);
191   }
192
193   D_Dictionnaire D_chargerDico (char nomFichier[]) {
194     assert(F_fichierExiste(nomFichier)==TRUE);
195
196     FILE *fichierSDDDico;
197     D_Dictionnaire dico;
198
199     fichierSDDDico=fopen(nomFichier, "r");
200     dico=D_creerDico();
201     deserialiser(&dico, fichierSDDDico);
202     fclose(fichierSDDDico);
203
204     return dico;
205   }
```

8.5 correcteurOrthographique.h

```
1
2
3   #ifndef __CORRECTEUR_ORTHOGRAPHIQUE__
4   #define __CORRECTEUR_ORTHOGRAPHIQUE__
5
6   #include "listeChaineeMot.h"
7   #include "mot.h"
8   #include "ensemble.h"
9   #include "dictionnaire.h"
10
11
12  typedef struct CO_CorrecteurOrthographique {
```

```
13    M_Mot mot;
14    D_Dictionnaire dico;
15  } CO_CorrecteurOrthographique;
16
17
18  CO_CorrecteurOrthographique CO_correcteurOrthographique();
19
20
21  M_Mot CO_obtenirMot(CO_CorrecteurOrthographique co);
22
23
24  D_Dictionnaire CO_obtenirDictionnaire(CO_CorrecteurOrthographique co);
25
26
27  void CO_fixerDictionnaire(CO_CorrecteurOrthographique* co, D_Dictionnaire dico);
28
29
30  void CO_fixerMot(CO_CorrecteurOrthographique* co, M_Mot mot);
31
32
33  E_Ensemble CO_proposerCorrections(CO_CorrecteurOrthographique co);
34
35
36  int CO_estBienOrthographie(CO_CorrecteurOrthographique co);
37
38  #endif
```

8.6 correcteurOrthographique.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <assert.h>
4  #include <string.h>
5  #include "correcteurOrthographique.h"
6  #include "listeChaineeMot.h"
7  #include "mot.h"
8  #include "ensemble.h"
9  #include "dictionnaire.h"
10
11 #define NDEBUG
12
13 #define LETTRES_SPECIALES_AUTORISEES "ÀÂÄÆÇÈÉÊËÌÎÏÒÓÔÖÙÚÛÜàâäæçèéêëìîïòôöùûü"
```

```
14 #define LONGUEUR_LETTRES_SPECIALES_AUTORISEES 38
15
16
17
18 void ajouterSiCorrecte(E_Ensemble* corrections, CO_CorrecteurOrthographique co)
19 {
20   if (CO_estBienOrthographie(co)) {
21     E_ajouter(corrections, CO_obtenirMot(co));
22   }
23 }
24
25
26 CO_CorrecteurOrthographique CO_correcteurOrthographique(){
27   CO_CorrecteurOrthographique co;
28   M_Mot mot;
29   D_Dictionnaire dico;
30
31   M_fixerLaChaine(&mot, "");
32   M_fixerLongueur(&mot, 0);
33   CO_fixerMot(&co, mot);
34
35   dico = NULL;
36   CO_fixerDictionnaire(&co, dico);
37
38   return co;
39 }
40
41 M_Mot CO_obtenirMot(CO_CorrecteurOrthographique co){
42   return co.mot;
43 }
44
45 D_Dictionnaire CO_obtenirDictionnaire(CO_CorrecteurOrthographique co){
46   return co.dico;
47 }
48
49 void CO_fixerDictionnaire(CO_CorrecteurOrthographique* co, D_Dictionnaire dico)
50 {
51   co->dico = dico;
52 }
53
54 void CO_fixerMot(CO_CorrecteurOrthographique* co, M_Mot mot){
55   co->mot = mot;
```

```
56  }
57
58  E_Ensemble CO_proposerCorrections(CO_CorrecteurOrthographique co) {
59    assert(!(CO_estBienOrthographie(co)));
60
61    E_Ensemble corrections;
62    M_Mot temp, mot;
63    D_Dictionnaire dico;
64    CO_CorrecteurOrthographique coTest;
65    unsigned int i, k;
66    char j;
67
68    coTest = CO_correcteurOrthographique();
69
70    mot = CO_obtenirMot(co);
71    dico = CO_obtenirDictionnaire(co);
72
73    CO_fixerDictionnaire(&coTest, dico);
74
75    corrections = E_ensemble();
76
77    if (M_obtenirLongueur(mot)>=2) {
78      for (i=0;i<M_obtenirLongueur(mot)-1;i++) {
79        temp = M_inverserDeuxLettresConsecutives(mot, i);
80        CO_fixerMot(&coTest, temp);
81        ajouterSiCorrecte(&corrections, coTest);
82      }
83
84      for (i=0;i<M_obtenirLongueur(mot);i++) {
85        temp = M_supprimerLettre(mot, i);
86        CO_fixerMot(&coTest, temp);
87        ajouterSiCorrecte(&corrections, coTest);
88      }
89    }
90
91    for (j='a';j<='z';j++) {
92      for (i=0;i<M_obtenirLongueur(mot);i++) {
93        temp = M_remplacerLettre(mot, i, j);
94        CO_fixerMot(&coTest, temp);
95        ajouterSiCorrecte(&corrections, coTest);
96      }
97    }
```

```
98
99    while (k<=LONGUEUR_LETTRES_SPECIALES_AUTORISEES-1){
100     for (i=0;i<M_obtenirLongueur(mot);i++) {
101       temp = M_remplacerLettre(mot, i, LETTRES_SPECIALES_AUTORISEES[k]);
102       CO_fixerMot(&coTest, temp);
103       ajouterSiCorrecte(&corrections, coTest);
104     }
105     k=k+1;
106   }
107
108   for (j='a';j<='z';j++) {
109     for (i=0;i<=M_obtenirLongueur(mot);i++) {
110       temp = M_insererLettre(mot, j, i);
111       CO_fixerMot(&coTest, temp);
112       ajouterSiCorrecte(&corrections, coTest);
113     }
114   }
115
116   while (k<=LONGUEUR_LETTRES_SPECIALES_AUTORISEES-1){
117     for (i=0;i<=M_obtenirLongueur(mot);i++) {
118       temp = M_insererLettre(mot, LETTRES_SPECIALES_AUTORISEES[k], i);
119       CO_fixerMot(&coTest, temp);
120       ajouterSiCorrecte(&corrections, coTest);
121     }
122     k=k+1;
123   }
124
125   return corrections;
126 }
127
128 int CO_estBienOrthographie(CO_CorrecteurOrthographique co) {
129   return D_estPresent(CO_obtenirDictionnaire(co), CO_obtenirMot(co));
130 }
```

8.7 arbreN.h

```
1
2
3 #ifndef __arbreN__
4 #define __arbreN__
5
6
```

```
7  typedef struct AbN_Noeud* AbN_ArbreN;
8
9
10 typedef struct AbN_Noeud {
11   char lettre;
12   int motValide;
13   AbN_ArbreN Fils;
14   AbN_ArbreN Frere;
15 } AbN_Noeud;
16
17
18 AbN_ArbreN AbN_creerArbreNonInit();
19
20
21 int AbN_estVide(AbN_ArbreN arbre);
22
23
24 int AbN_obtenirBool (AbN_ArbreN arbre);
25
26
27 char AbN_obtenirChar (AbN_ArbreN arbre);
28
29
30 void AbN_fixerBool (AbN_ArbreN* arbre, int valide);
31
32
33 void AbN_fixerChar (AbN_ArbreN* arbre, char lalettre);
34
35
36 AbN_ArbreN AbN_obtenirFils (AbN_ArbreN arbre);
37
38
39 AbN_ArbreN AbN_obtenirFrere (AbN_ArbreN arbre);
40
41
42 void AbN_fixerFils (AbN_ArbreN* arbre, AbN_ArbreN fils);
43
44
45 void AbN_fixerFrere (AbN_ArbreN* arbre, AbN_ArbreN frere);
46
47 #endif
```

8.8 arbreN.c

```
1  #include <stdlib.h>
2  #include <assert.h>
3  #include "arbreN.h"
4  #define NDEBUG
5
6  #define TRUE 1
7  #define FALSE 0
8  #define charnul '\0'
9
10 AbN_ArbreN AbN_creerArbreNonInit (){
11    AbN_ArbreN temp= (AbN_ArbreN)malloc(sizeof(AbN_Noeud));
12    AbN_fixerFils(&temp, NULL);
13    AbN_fixerFrere(&temp, NULL);
14    AbN_fixerChar (&temp, charnul);
15    AbN_fixerBool (&temp, 0);
16
17    return temp;
18 }
19
20 int AbN_estVide(AbN_ArbreN arbre){
21    return (arbre==NULL);
22 }
23
24 int AbN_obtenirBool (AbN_ArbreN arbre){
25    assert(!AbN_estVide(arbre));
26    return (*arbre).motValide;
27 }
28
29 char AbN_obtenirChar (AbN_ArbreN arbre){
30    assert(!AbN_estVide(arbre));
31    return (*arbre).lettre;
32 }
33
34 void AbN_fixerBool (AbN_ArbreN* arbre, int valide){
35    assert(!AbN_estVide(*arbre));
36    (*arbre)->motValide=valide;
37 }
38
39 void AbN_fixerChar (AbN_ArbreN* arbre, char lalettre){
40    assert(!AbN_estVide(*arbre));
```

```
41    (*arbre)->lettre=lalettre;
42  }
43
44  AbN_ArbreN AbN_obtenirFils (AbN_ArbreN arbre){
45    assert(!AbN_estVide(arbre));
46    return (*arbre).Fils;
47  }
48
49  AbN_ArbreN AbN_obtenirFrere (AbN_ArbreN arbre){
50    assert(!AbN_estVide(arbre));
51    return (*arbre).Frere;
52  }
53
54  void AbN_fixerFils (AbN_ArbreN* arbre, AbN_ArbreN fils) {
55    assert(!AbN_estVide(*arbre));
56    (*arbre)->Fils=fils;
57  }
58
59  void AbN_fixerFrere (AbN_ArbreN* arbre, AbN_ArbreN frere) {
60    assert(!AbN_estVide(*arbre));
61    (*arbre)->Frere=frere;
62  }
```

8.9 ensemble.h

```
1
2
3   #ifndef __ENSEMBLE__
4   #define __ENSEMBLE__
5
6   #include "listeChaineeMot.h"
7   #include "mot.h"
8
9
10  typedef M_Mot E_Element;
11
12
13  typedef struct E_Ensemble {
14    LCM_ListeChaineeMot lesElements;
15    unsigned int nbElement;
16  } E_Ensemble;
17
```

```
18
19 LCM_ListeChaineeMot E_obtenirLesElements (E_Ensemble e);
20
21
22 unsigned int E_obtenirNbElement (E_Ensemble e);
23
24
25 void E_fixerLesElements (E_Ensemble* e, LCM_ListeChaineeMot listeElements);
26
27
28 E_Ensemble E_ensemble();
29
30
31 void E_ajouter(E_Ensemble* e, E_Element el);
32
33
34 unsigned int E_cardinalite(E_Ensemble e);
35
36
37 int E_estPresent(E_Ensemble e, E_Element el);
38
39
40 void E_retirer(E_Ensemble* e, E_Element el);
41
42 #endif
```

8.10 ensemble.c

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include "ensemble.h"
4 #include "listeChaineeMot.h"
5 #include "mot.h"
6
7 #define TRUE 1
8 #define FALSE 0
9
10
11 int estPresentDansLesElements(LCM_ListeChaineeMot l, M_Mot mot) {
12   if (LCM_estVide(l)) {
13     return FALSE;
14   }
```

```
15    else {
16      if (M_sontEgaux(LCM_obtenirElement(l), mot)) {
17        return TRUE;
18      }
19      else {
20        return estPresentDansLesElements(LCM_obtenirListeSuivante(l), mot);
21      }
22    }
23  }
24
25  void retirerSiPresent(LCM_ListeChaineeMot l, M_Mot mot, int *etaitPresent){
26    LCM_ListeChaineeMot temp;
27
28    if (LCM_estVide(l)) {
29      *etaitPresent=FALSE;
30    }
31    else {
32      if (M_sontEgaux(LCM_obtenirElement(l), mot)) {
33        LCM_supprimerTete(&l);
34        *etaitPresent=TRUE;
35      }
36      else {
37        temp=LCM_obtenirListeSuivante(l);
38        retirerSiPresent(temp, mot, etaitPresent);
39        LCM_fixerListeSuivante(l, temp);
40      }
41    }
42  }
43
44
45  LCM_ListeChaineeMot E_obtenirLesElements (E_Ensemble e) {
46    return e.lesElements;
47  }
48
49  unsigned int E_obtenirNbElement (E_Ensemble e) {
50    return e.nbElement;
51  }
52
53  void E_fixerLesElements (E_Ensemble *e, LCM_ListeChaineeMot listeElements) {
54    LCM_fixerListeSuivante(e->lesElements, listeElements);
55  }
56
```

```
57 E_Ensemble E_ensemble(){
58   E_Ensemble resultat;
59
60   resultat.lesElements = LCM_listeChaineeMot();
61   resultat.nbElement = 0;
62
63   return resultat;
64 }
65
66 void E_ajouter(E_Ensemble* e, E_Element el){
67   if (!(E_estPresent(*e, el))) {
68     LCM_ajouter(&e->lesElements,el);
69     e->nbElement++;
70   }
71 }
72
73 unsigned int E_cardinalite(E_Ensemble e){
74   return E_obtenirNbElement(e);
75 }
76
77 int E_estPresent(E_Ensemble e, E_Element el) {
78   return estPresentDansLesElements(E_obtenirLesElements(e), el);
79 }
80
81 void E_retirer(E_Ensemble *e, E_Element el){
82   int etaitPresent;
83
84   retirerSiPresent(E_obtenirLesElements(*e), el, &etaitPresent);
85
86   if (etaitPresent) {
87     e->nbElement--;
88   }
89 }
```

8.11 listeChaineeMot.h

```
1
2
3 #ifndef _LISTE_CHAINEE_MOT_
4 #define _LISTE_CHAINEE_MOT_
5
6 #include <errno.h>
```

```
7  #include "mot.h"
8
9
10  #define LCM_ERREUR_MEMOIRE 1
11
12
13 typedef struct LCM_Noeud *LCM_ListeChaineeMot;
14
15
16 typedef struct LCM_Noeud {
17         M_Mot element;
18         LCM_ListeChaineeMot listeSuivante;
19 } LCM_Noeud;
20
21
22 LCM_ListeChaineeMot LCM_listeChaineeMot();
23
24
25 int LCM_estVide(LCM_ListeChaineeMot l);
26
27
28 void LCM_ajouter(LCM_ListeChaineeMot *pl, M_Mot mot);
29
30
31 M_Mot LCM_obtenirElement(LCM_ListeChaineeMot l);
32
33
34 LCM_ListeChaineeMot LCM_obtenirListeSuivante(LCM_ListeChaineeMot l);
35
36
37 void LCM_fixerListeSuivante(LCM_ListeChaineeMot pl, LCM_ListeChaineeMot l);
38
39
40 void LCM_fixerElement(LCM_ListeChaineeMot pl, M_Mot mot);
41
42
43 void LCM_supprimerTete(LCM_ListeChaineeMot *pl);
44
45
46 void LCM_supprimer(LCM_ListeChaineeMot *pl);
47
48 #endif
```

8.12 listeChaineeMot.c

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4  #include <assert.h>
5  #include <stdbool.h>
6  #include "listeChaineeMot.h"
7
8  #define NDEBUG
9
10 #define TRUE 1
11 #define FALSE 0
12
13 LCM_ListeChaineeMot LCM_listeChaineeMot() {
14         errno=0;
15         return NULL;
16 }
17
18 int LCM_estVide(LCM_ListeChaineeMot l) {
19         errno=0;
20         return (l==NULL);
21 }
22
23 void LCM_ajouter(LCM_ListeChaineeMot *pl, M_Mot mot) {
24         LCM_ListeChaineeMot pNoeud=(LCM_ListeChaineeMot)malloc(sizeof(
25                                                                 LCM_Noeud));
26         if (pNoeud!=NULL) {
27                 errno=0;
28                 pNoeud->element=mot;
29                 pNoeud->listeSuivante=*pl;
30                 *pl=pNoeud;
31         }
32         else {
33                 errno=LCM_ERREUR_MEMOIRE;
34         }
35 }
36
37 M_Mot LCM_obtenirElement(LCM_ListeChaineeMot l) {
38         assert(!LCM_estVide(l));
39         errno=0;
40         return l->element;
```

```
41 }
42
43 LCM_ListeChaineeMot LCM_obtenirListeSuivante(LCM_ListeChaineeMot l) {
44         assert (!LCM_estVide(l));
45         errno=0;
46         return l->listeSuivante;
47 }
48
49 void LCM_fixerListeSuivante(LCM_ListeChaineeMot pl,
50                             LCM_ListeChaineeMot suivant) {
51         assert (!LCM_estVide(pl));
52         errno=0;
53         pl->listeSuivante=suivant;
54 }
55
56 void LCM_fixerElement(LCM_ListeChaineeMot pl, M_Mot mot) {
57         assert (!LCM_estVide(pl));
58         errno=0;
59         pl->element=mot;
60 }
61
62 void LCM_supprimerTete(LCM_ListeChaineeMot *pl) {
63         assert (!LCM_estVide(*pl));
64
65         LCM_ListeChaineeMot temp;
66
67         errno=0;
68         temp=*pl;
69         *pl=LCM_obtenirListeSuivante(*pl);
70         free(temp);
71 }
72
73 void LCM_supprimer(LCM_ListeChaineeMot *pl) {
74         errno=0;
75         if (!LCM_estVide(*pl)) {
76                 LCM_supprimerTete(pl);
77                 LCM_supprimer(pl);
78         }
79 }
```

8.13 affichages.h

```
1
2
3  #ifndef __affichages__
4  #define __affichages__
5
6  #include "ensemble.h"
7
8
9  void A_afficherCorrectionEtPosMot(E_Ensemble lesCorrections, M_Mot leMot, unsigned
10
11
12 void A_affichageMotBienEcrit();
13
14
15 void A_afficherAide();
16
17 #endif
```

8.14 affichages.c

```
1  #include <stdio.h>
2  #include "ensemble.h"
3
4  #define TAILLE_MAX_MOT_FICHIER_AIDE 150
5  #define NOM_FICHIER_AIDE "./fichierAide.txt"
6
7  void A_afficherCorrectionEtPosMot(E_Ensemble lesCorrections, M_Mot leMot,
8                                    unsigned int pos) {
9    unsigned int i, nbCorrections;
10   M_Mot mot;
11   LCM_ListeChaineeMot lesMotsCorrects;
12
13   lesMotsCorrects = E_obtenirLesElements(lesCorrections);
14   nbCorrections=E_cardinalite(lesCorrections);
15
16   printf("& %s %d %d: ", M_obtenirLaChaine(leMot), nbCorrections, pos);
17
18   for (i=0; i<nbCorrections; i++) {
19     mot= LCM_obtenirElement(lesMotsCorrects);
20     printf("%s ", M_obtenirLaChaine(mot));
21     lesMotsCorrects = LCM_obtenirListeSuivante(lesMotsCorrects);
22   }
```

```
23
24    printf("\n");
25  }
26
27  void A_affichageMotBienEcrit() {
28    printf("*\n");
29  }
30
31  void A_afficherAide() {
32    FILE* fichierAide=NULL;
33    char chaine[TAILLE_MAX_MOT_FICHIER_AIDE];
34
35    fichierAide=fopen(NOM_FICHIER_AIDE, "r");
36
37    if (fichierAide != NULL) {
38      while (fgets(chaine, TAILLE_MAX_MOT_FICHIER_AIDE, fichierAide) != NULL) {
39        printf("%s", chaine);
40      }
41
42      fclose(fichierAide);
43    }
44    else {
45      printf("Le fichier d'aide est introuvable\n");
46    }
47  }
```

8.15 existeFichier.h

```
1
2
3  #ifndef __existeFichier__
4  #define __existeFichier__
5
6
7  int F_fichierExiste (char *nomFichier);
8
9  #endif
```

8.16 existeFichier.c

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

```
2
3  #define TRUE 1
4  #define FALSE 0
5
6  int F_fichierExiste (char *nomFichier) {
7    FILE* fichier=NULL;
8
9    fichier=fopen(nomFichier, "r");
10
11
12   if (fichier != NULL) {
13     fclose(fichier);
14     return TRUE;
15   }
16   else {
17     return FALSE;
18   }
19 }
```

8.17 main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "affichages.h"
5  #include "arbreN.h"
6  #include "correcteurOrthographique.h"
7  #include "dictionnaire.h"
8  #include "ensemble.h"
9  #include "existeFichier.h"
10 #include "listeChaineeMot.h"
11 #include "mot.h"
12
13 #define TRUE 1
14 #define FALSE 0
15
16 void mettreMotsDansSDDdico (char *nomFichierSDDDico,
17                             char *nomFichierTexteMotsAInserer) {
18   FILE* fichierTexteMotsAInserer = NULL;
19   M_Mot mot;
20   D_Dictionnaire SDDDico, fils;
21   char temp[TAILLE_MAX_MOT];
```

```
22    unsigned int longueurMot;
23
24    SDDDico=D_creerDico();
25
26    if (F_fichierExiste(nomFichierSDDDico)) {
27      fils=D_chargerDico(nomFichierSDDDico);
28      AbN_fixerFils(&SDDDico, fils);
29
30    }
31    else {
32      fils=AbN_obtenirFils(SDDDico);
33    }
34
35    fichierTexteMotsAInserer=fopen(nomFichierTexteMotsAInserer, "r");
36
37    while (fgets(temp, TAILLE_MAX_MOT, fichierTexteMotsAInserer) != NULL) {
38      longueurMot=(unsigned int)strlen(temp)-1;
39      temp[longueurMot]='\0';
40      M_fixerLongueur(&mot, longueurMot);
41      M_fixerLaChaine(&mot, temp);
42
43      if (!(D_estPresent(fils, mot))) {
44        D_insererMot(&fils, mot);
45      }
46    }
47    fclose(fichierTexteMotsAInserer);
48
49    AbN_fixerFils(&SDDDico, fils);
50    D_sauvegarderDico(nomFichierSDDDico, fils);
51
52    free(fils);
53    free(SDDDico);
54  }
55
56  unsigned int queFaireEnFonctionDeCommandeEntree (int argc, char **argv) {
57    unsigned int nbArgument;
58    unsigned int resultat;
59
60    nbArgument=argc-1;
61    resultat=1;
62
63    if ((nbArgument==2) | (nbArgument==4)) {
```

```
64      if (strcmp(argv[1], "-d")==0) {
65
66        if (nbArgument==4) {
67          if ((strcmp(argv[3], "-f")==0) & (F_fichierExiste(argv[4]))) {
68            resultat=3;
69          }
70        }
71        else if (F_fichierExiste(argv[2])) {
72          resultat=2;
73        }
74      }
75    }
76    return resultat;
77  }
78
79
80  void corrigerMot(CO_CorrecteurOrthographique co, unsigned int posMotFaux) {
81    E_Ensemble correctionsPossibles;
82    LCM_ListeChaineeMot temp;
83
84    correctionsPossibles = E_ensemble();
85
86    if (!(CO_estBienOrthographie(co))) {
87      correctionsPossibles = CO_proposerCorrections(co);
88      A_afficherCorrectionEtPosMot(correctionsPossibles, CO_obtenirMot(co),
89                                   posMotFaux);
90    }
91    else {
92      A_affichageMotBienEcrit();
93    }
94
95    temp=E_obtenirLesElements(correctionsPossibles);
96    free(temp);
97  }
98
99  int main(int argc,char** argv) {
100   unsigned int actionAFaire;
101   int posDepuisDebutFlux;
102   M_Mot mot;
103   int arret=FALSE;
104   D_Dictionnaire SDDDico, fils;
105   CO_CorrecteurOrthographique co;
```

```
106
107     actionAFaire=queFaireEnFonctionDeCommandeEntree(argc, argv);
108
109     if (actionAFaire==1) {
110       A_afficherAide();
111     }
112
113     if (actionAFaire==2) {
114       SDDDico = D_creerDico();
115       fils = D_chargerDico(argv[2]);
116       AbN_fixerFils(&SDDDico, fils);
117       co = CO_correcteurOrthographique();
118       CO_fixerDictionnaire(&co, fils);
119       M_fixerLongueur(&mot, 0);
120       posDepuisDebutFlux=-1;
121
122       while (arret==FALSE) {
123         M_obtenirMotEntreeStandard(&posDepuisDebutFlux, &mot, &arret);
124
125         if (arret==FALSE) {
126           CO_fixerMot(&co, mot);
127           corrigerMot(co, (unsigned int)posDepuisDebutFlux);
128         }
129       }
130       free(fils);
131       free(SDDDico);
132     }
133
134     if (actionAFaire==3) {
135       mettreMotsDansSDDdico(argv[2], argv[4]);
136     }
137     return 0;
138 }
```

9 Tests unitaires

9.1 motTU.c

```
1  #include <stdlib.h>
2  #include <CUnit/Basic.h>
3  #include <string.h>
4  #include "mot.h"
5
6  #define TRUE 1
7  #define FALSE 0
8
9  int init_suite_success(void) {
10   return 0;
11 }
12
13 int clean_suite_success(void) {
14   return 0;
15 }
16
17 void test_M_longueur(void) {
18   M_Mot mot;
19
20   M_fixerLaChaine(&mot, "");
21   CU_ASSERT_TRUE(M_longueur(mot)==0);
22   M_fixerLaChaine(&mot, "1");
23   CU_ASSERT_TRUE(M_longueur(mot)==1);
24   M_fixerLaChaine(&mot, "12345678");
25   CU_ASSERT_TRUE(M_longueur(mot)==8);
26   M_fixerLaChaine(&mot, "écrire");
27   CU_ASSERT_TRUE(M_longueur(mot)==6);
28 }
29
30 void test_M_obtenirIemeCaractere(void) {
31   M_Mot mot;
32
33   M_fixerLaChaine(&mot, "bonsoir");
34   M_fixerLongueur(&mot, 7);
35   CU_ASSERT_TRUE(M_obtenirIemeCaractere(mot, 0)=='b');
36   CU_ASSERT_TRUE(M_obtenirIemeCaractere(mot, 3)=='s');
37   CU_ASSERT_TRUE(M_obtenirIemeCaractere(mot, 6)=='r');
38
```

```
39    M_fixerLaChaine(&mot, "écrire");
40    M_fixerLongueur(&mot, 6);
41    CU_ASSERT_TRUE(M_obtenirIemeCaractere(mot, 0)=='é');
42    CU_ASSERT_TRUE(M_obtenirIemeCaractere(mot, 3)=='i');
43    CU_ASSERT_TRUE(M_obtenirIemeCaractere(mot, 5)=='e');
44  }
45
46  void test_M_remplacerLettre(void) {
47    M_Mot mot;
48    M_Mot temp;
49
50    M_fixerLaChaine(&mot, "bonsoir");
51    M_fixerLongueur(&mot, 7);
52
53    temp=M_remplacerLettre(mot, 0, 'a');
54    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "aonsoir")==0);
55    temp=M_remplacerLettre(mot, 3, 'a');
56    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonaoir")==0);
57    temp=M_remplacerLettre(mot, 6, 'a');
58    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonsoia")==0);
59
60    M_fixerLaChaine(&mot, "écrire");
61    M_fixerLongueur(&mot, 6);
62
63    temp=M_remplacerLettre(mot, 0, 'a');
64    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "acrire")==0);
65    temp=M_remplacerLettre(mot, 3, 'a');
66    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrare")==0);
67    temp=M_remplacerLettre(mot, 5, 'a');
68    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrira")==0);
69  }
70
71  void test_M_inverserDeuxLettresConsecutives(void) {
72    M_Mot mot;
73    M_Mot temp;
74
75    M_fixerLaChaine(&mot, "bonsoir");
76    M_fixerLongueur(&mot, 7);
77
78    temp=M_inverserDeuxLettresConsecutives(mot, 0);
79    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "obnsoir")==0);
80    temp=M_inverserDeuxLettresConsecutives(mot, 3);
```

```
81    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonosir")==0);
82    temp=M_inverserDeuxLettresConsecutives(mot, 5);
83    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonsori")==0);
84
85    M_fixerLaChaine(&mot, "écrire");
86    M_fixerLongueur(&mot, 6);
87
88    temp=M_inverserDeuxLettresConsecutives(mot, 0);
89    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "cérire")==0);
90    temp=M_inverserDeuxLettresConsecutives(mot, 3);
91    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrrie")==0);
92    temp=M_inverserDeuxLettresConsecutives(mot, 4);
93    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrier")==0);
94  }
95
96  void test_M_supprimerLettre(void) {
97    M_Mot mot;
98    M_Mot temp;
99
100   M_fixerLaChaine(&mot, "bonsoir");
101   M_fixerLongueur(&mot, 7);
102
103   temp=M_supprimerLettre(mot, 0);
104   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "onsoir")==0);
105   CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==6);
106   temp=M_supprimerLettre(mot, 3);
107   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonoir")==0);
108   CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==6);
109   temp=M_supprimerLettre(mot, 6);
110   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonsoi")==0);
111   CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==6);
112
113   M_fixerLaChaine(&mot, "écrire");
114   M_fixerLongueur(&mot, 6);
115
116   temp=M_supprimerLettre(mot, 0);
117   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "crire")==0);
118   CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==5);
119   temp=M_supprimerLettre(mot, 3);
120   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrre")==0);
121   CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==5);
122   temp=M_supprimerLettre(mot, 5);
```

```
123     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrir")==0);
124     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==5);
125   }
126
127   void test_M_insererLettre(void) {
128     M_Mot mot;
129     M_Mot temp;
130
131     M_fixerLaChaine(&mot, "bonsoir");
132     M_fixerLongueur(&mot, 7);
133
134     temp=M_insererLettre(mot, 'a', 0);
135     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "abonsoir")==0);
136     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==8);
137     temp=M_insererLettre(mot, 'a', 3);
138     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonasoir")==0);
139     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==8);
140     temp=M_insererLettre(mot, 'a', 6);
141     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonsoiar")==0);
142     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==8);
143     temp=M_insererLettre(mot, 'a', 7);
144     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonsoira")==0);
145     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==8);
146
147     M_fixerLaChaine(&mot, "écrire");
148     M_fixerLongueur(&mot, 6);
149
150     temp=M_insererLettre(mot, 'a', 0);
151     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "aécrire")==0);
152     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==7);
153     temp=M_insererLettre(mot, 'a', 3);
154     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écraire")==0);
155     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==7);
156     temp=M_insererLettre(mot, 'a', 5);
157     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrirae")==0);
158     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==7);
159     temp=M_insererLettre(mot, 'a', 6);
160     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrirea")==0);
161     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(temp))==7);
162   }
163
164   void test_M_fixerIemeCaractere(void) {
```

```
165     M_Mot mot;
166     M_Mot temp;
167 
168     M_fixerLaChaine(&mot, "bonsoir");
169     M_fixerLongueur(&mot, 7);
170 
171     temp=mot;
172     M_fixerIemeCaractere(&temp, 0, 'a');
173     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "aonsoir")==0);
174 
175     temp=mot;
176     M_fixerIemeCaractere(&temp, 3, 'a');
177     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonaoir")==0);
178 
179     temp=mot;
180     M_fixerIemeCaractere(&temp, 6, 'a');
181     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "bonsoia")==0);
182 
183     M_fixerLaChaine(&mot, "écrire");
184     M_fixerLongueur(&mot, 6);
185 
186     temp=mot;
187     M_fixerIemeCaractere(&temp, 0, 'a');
188     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "acrire")==0);
189 
190     temp=mot;
191     M_fixerIemeCaractere(&temp, 3, 'a');
192     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrare")==0);
193 
194     temp=mot;
195     M_fixerIemeCaractere(&temp, 5, 'a');
196     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(temp), "écrira")==0);
197 }
198 
199 void test_M_fixerLaChaine(void) {
200     M_Mot mot;
201     char temp[10];
202 
203     M_fixerLaChaine (&mot, "bonsoir");
204     M_fixerLongueur(&mot, 7);
205     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(mot), "bonsoir")==0);
206     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(mot))==7);
```

```
207     strcpy(temp, M_obtenirLaChaine(mot));
208
209     CU_ASSERT_TRUE(temp[7]=='\0');
210
211     M_fixerLaChaine (&mot, "écrire");
212     M_fixerLongueur(&mot, 6);
213     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(mot), "écrire")==0);
214     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(mot))==6);
215     strcpy(temp, M_obtenirLaChaine(mot));
216     CU_ASSERT_TRUE(temp[6]=='\0');
217
218 }
219
220 void test_M_obtenirLaChaine(void) {
221     M_Mot mot;
222     char temp[10];
223
224     M_fixerLaChaine(&mot, "bonsoir");
225     M_fixerLongueur(&mot, 7);
226     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(mot), "bonsoir")==0);
227     CU_ASSERT_TRUE(strlen(M_obtenirLaChaine(mot))==7);
228     strcpy(temp, M_obtenirLaChaine(mot));
229     CU_ASSERT_TRUE(temp[7]=='\0');
230
231     M_fixerLaChaine(&mot, "");
232     M_fixerLongueur(&mot, 0);
233     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(mot), "")==0);
234     strcpy(temp, M_obtenirLaChaine(mot));
235
236     CU_ASSERT_TRUE(temp[0]=='\0');
237 }
238
239 void test_M_fixerLongueur(void) {
240     M_Mot mot;
241
242     M_fixerLaChaine(&mot, "bonsoir");
243     M_fixerLongueur(&mot, 7);
244     CU_ASSERT_TRUE(M_obtenirLongueur(mot)==strlen(M_obtenirLaChaine(mot)));
245
246     M_fixerLaChaine(&mot, "écrire");
247     M_fixerLongueur(&mot, 6);
248     CU_ASSERT_TRUE(M_obtenirLongueur(mot)==strlen(M_obtenirLaChaine(mot)));
```

```
249 }
250
251 void test_M_obtenirLongueur(void) {
252   M_Mot mot;
253
254   M_fixerLaChaine(&mot, "bonsoir");
255   M_fixerLongueur(&mot, 7);
256   CU_ASSERT_TRUE(M_obtenirLongueur(mot)==strlen(M_obtenirLaChaine(mot)));
257
258   M_fixerLaChaine(&mot, "écrire");
259   M_fixerLongueur(&mot, 6);
260   CU_ASSERT_TRUE(M_obtenirLongueur(mot)==strlen(M_obtenirLaChaine(mot)));
261 }
262
263 void test_M_sontEgaux(void) {
264   M_Mot mot1, mot2;
265
266   M_fixerLaChaine(&mot1, "bonsoir");
267   M_fixerLongueur(&mot1, 7);
268   M_fixerLaChaine(&mot2, "bonsoir");
269   M_fixerLongueur(&mot2, 7);
270   CU_ASSERT_TRUE(M_sontEgaux(mot1, mot2)==TRUE);
271
272   M_fixerLaChaine(&mot2, "bonsoi");
273   M_fixerLongueur(&mot2, 6);
274   CU_ASSERT_TRUE(M_sontEgaux(mot1, mot2)==FALSE);
275
276
277   M_fixerLaChaine(&mot2, "bonjour");
278   M_fixerLongueur(&mot2, 7);
279   CU_ASSERT_TRUE(M_sontEgaux(mot1, mot2)==FALSE);
280
281   M_fixerLaChaine(&mot1, "");
282   M_fixerLongueur(&mot1, 0);
283   M_fixerLaChaine(&mot2, "");
284   M_fixerLongueur(&mot2, 0);
285   CU_ASSERT_TRUE(M_sontEgaux(mot1, mot2)==TRUE);
286
287   M_fixerLongueur(&mot2, 1);
288   CU_ASSERT_TRUE(M_sontEgaux(mot1, mot2)==FALSE);
289
290   M_fixerLaChaine(&mot1, "écrire");
```

```
291     M_fixerLongueur(&mot1, 6);
292     M_fixerLaChaine(&mot2, "écrire");
293     M_fixerLongueur(&mot2, 6);
294     CU_ASSERT_TRUE(M_sontEgaux(mot1, mot2)==TRUE);
295   }
296
297   void test_M_estUneLettre(void) {
298
299     CU_ASSERT_TRUE(M_estUneLettre('e')==TRUE);
300     CU_ASSERT_TRUE(M_estUneLettre('E')==TRUE);
301     CU_ASSERT_TRUE(M_estUneLettre('%')==FALSE);
302     CU_ASSERT_TRUE(M_estUneLettre('!')==FALSE);
303   }
304
305   int main(int argc, char** argv){
306     CU_pSuite pSuite = NULL;
307
308
309     if (CUE_SUCCESS != CU_initialize_registry())
310       return CU_get_error();
311
312
313     pSuite = CU_add_suite("Tests␣boite␣noire␣:␣MotTU.c", init_suite_success,
314                           clean_suite_success);
315     if (NULL == pSuite) {
316       CU_cleanup_registry();
317       return CU_get_error();
318     }
319
320
321     if ((NULL == CU_add_test(pSuite, "M_longueur", test_M_longueur))
322         || (NULL == CU_add_test(pSuite, "M_obtenirIemeCaractere",
323                                 test_M_obtenirIemeCaractere))
324         || (NULL == CU_add_test(pSuite, "M_remplacerLettre",
325                                 test_M_remplacerLettre))
326         || (NULL == CU_add_test(pSuite, "M_inverserDeuxLettresConsecutives",
327                                 test_M_inverserDeuxLettresConsecutives))
328         || (NULL == CU_add_test(pSuite, "M_supprimerLettre",
329                                 test_M_supprimerLettre))
330         || (NULL == CU_add_test(pSuite, "M_insererLettre", test_M_insererLettre))
331         || (NULL == CU_add_test(pSuite, "M_fixerIemeCaractere",
332                                 test_M_fixerIemeCaractere))
```

```
333         || (NULL == CU_add_test(pSuite, "M_fixerLaChaine", test_M_fixerLaChaine))
334         || (NULL == CU_add_test(pSuite, "M_obtenirLaChaine",
335                                 test_M_obtenirLaChaine))
336         || (NULL == CU_add_test(pSuite, "M_fixerLongueur", test_M_fixerLongueur))
337         || (NULL == CU_add_test(pSuite, "M_obtenirLongueur",
338                                 test_M_obtenirLongueur))
339         || (NULL == CU_add_test(pSuite, "M_sontEgaux", test_M_sontEgaux))
340         || (NULL == CU_add_test(pSuite, "M_estUneLettre", test_M_estUneLettre))
341       )
342     {
343       CU_cleanup_registry();
344       return CU_get_error();
345     }
346
347
348   CU_basic_set_mode(CU_BRM_VERBOSE);
349   CU_basic_run_tests();
350   printf("\n");
351   CU_basic_show_failures(CU_get_failure_list());
352   printf("\n\n");
353
354
355   CU_cleanup_registry();
356   return CU_get_error();
357 }
```

9.2 dictionnaireTU.c

```
1  #include <stdlib.h>
2  #include <CUnit/Basic.h>
3  #include <string.h>
4  #include "mot.h"
5  #include "dictionnaire.h"
6  #include "existeFichier.h"
7  #include "arbreN.h"
8
9  #define TRUE 1
10 #define FALSE 0
11
12 int init_suite_success(void) {
13   return 0;
14 }
```

```
15
16 int clean_suite_success(void) {
17   return 0;
18 }
19
20 void test_D_estVide(void) {
21   D_Dictionnaire dico;
22
23   dico = D_creerDico();
24   CU_ASSERT_TRUE(D_estVide(dico)==FALSE);
25 }
26
27 void test_D_estPresent(void) {
28   D_Dictionnaire dico, fils;
29   M_Mot mot1, mot2, mot3, mot4, mot5;
30
31   dico=D_creerDico();
32
33   strcpy(mot1.laChaine, "bonjour");
34   mot1.longueur = 7;
35   strcpy(mot2.laChaine, "bonsoir");
36   mot2.longueur = 7;
37   strcpy(mot3.laChaine, "yoyo");
38   mot3.longueur = 4;
39   strcpy(mot4.laChaine, "test");
40   mot4.longueur = 4;
41   strcpy(mot5.laChaine, "écrire");
42   mot5.longueur = 6;
43
44   fils = D_chargerDico("dicoTest.txt");
45   AbN_fixerFils(&dico, fils);
46
47   CU_ASSERT_TRUE(D_estPresent(fils, mot1));
48   CU_ASSERT_TRUE(D_estPresent(fils, mot2));
49   CU_ASSERT_TRUE(D_estPresent(fils, mot3));
50   CU_ASSERT_TRUE(D_estPresent(fils, mot4));
51   CU_ASSERT_TRUE(D_estPresent(fils, mot5));
52
53   free(fils);
54   free(dico);
55 }
56
```

```
57  void test_D_insererMot(void) {
58    D_Dictionnaire dico, fils;
59    M_Mot mot1, mot2, mot3, mot4, mot5, mot6, mot7, mot8, mot9, mot10, mot11,
60       mot12, mot13, mot14, mot15, mot16;
61
62    dico=D_creerDico();
63    fils=AbN_obtenirFils(dico);
64
65    strcpy(mot1.laChaine, "bonjour");
66    mot1.longueur = 7;
67    strcpy(mot2.laChaine, "bonsoir");
68    mot2.longueur = 7;
69    strcpy(mot3.laChaine, "yoyo");
70    mot3.longueur = 4;
71    strcpy(mot4.laChaine, "test");
72    mot4.longueur = 4;
73    strcpy(mot5.laChaine, "petit");
74    mot5.longueur = 5;
75    strcpy(mot6.laChaine, "correction");
76    mot6.longueur = 10;
77    strcpy(mot7.laChaine, "orthographique");
78    mot7.longueur = 14;
79    strcpy(mot8.laChaine, "un");
80    mot8.longueur = 2;
81    strcpy(mot9.laChaine, "de");
82    mot9.longueur = 2;
83    strcpy(mot10.laChaine, "avec");
84    mot10.longueur = 4;
85    strcpy(mot11.laChaine, "quelques");
86    mot11.longueur = 8;
87    strcpy(mot13.laChaine, "quelque");
88    mot13.longueur = 7;
89    strcpy(mot12.laChaine, "fautes");
90    mot12.longueur = 6;
91    strcpy(mot13.laChaine, "d");
92    mot13.longueur = 1;
93    strcpy(mot14.laChaine, "orthographe");
94    mot14.longueur = 11;
95    strcpy(mot15.laChaine, "écrire");
96    mot15.longueur = 6;
97    strcpy(mot16.laChaine, "petit-beure");
98    mot16.longueur = 11;
```

```
 99
100     D_insererMot(&fils, mot1);
101     D_insererMot(&fils, mot2);
102     D_insererMot(&fils, mot3);
103     D_insererMot(&fils, mot4);
104     D_insererMot(&fils, mot5);
105     D_insererMot(&fils, mot6);
106     D_insererMot(&fils, mot7);
107     D_insererMot(&fils, mot8);
108     D_insererMot(&fils, mot9);
109     D_insererMot(&fils, mot10);
110     D_insererMot(&fils, mot11);
111     D_insererMot(&fils, mot12);
112     D_insererMot(&fils, mot15);
113     D_insererMot(&fils, mot13);
114     D_insererMot(&fils, mot14);
115
116     AbN_fixerFils(&dico, fils);
117
118     CU_ASSERT_TRUE(D_estPresent(fils, mot1)==TRUE);
119     CU_ASSERT_TRUE(D_estPresent(fils, mot2)==TRUE);
120     CU_ASSERT_TRUE(D_estPresent(fils, mot3)==TRUE);
121     CU_ASSERT_TRUE(D_estPresent(fils, mot4)==TRUE);
122     CU_ASSERT_TRUE(D_estPresent(fils, mot5)==TRUE);
123     CU_ASSERT_TRUE(D_estPresent(fils, mot6)==TRUE);
124     CU_ASSERT_TRUE(D_estPresent(fils, mot7)==TRUE);
125     CU_ASSERT_TRUE(D_estPresent(fils, mot8)==TRUE);
126     CU_ASSERT_TRUE(D_estPresent(fils, mot9)==TRUE);
127     CU_ASSERT_TRUE(D_estPresent(fils, mot10)==TRUE);
128     CU_ASSERT_TRUE(D_estPresent(fils, mot11)==TRUE);
129     CU_ASSERT_TRUE(D_estPresent(fils, mot12)==TRUE);
130     CU_ASSERT_TRUE(D_estPresent(fils, mot13)==TRUE);
131     CU_ASSERT_TRUE(D_estPresent(fils, mot14)==TRUE);
132     CU_ASSERT_TRUE(D_estPresent(fils, mot15)==TRUE);
133
134     free(fils);
135     free(dico);
136 }
137
138 void test_D_chargerDico(void) {
139     D_Dictionnaire dico, fils;
140     M_Mot mot1, mot2, mot3, mot4, mot5;
```

```
141
142     dico=D_creerDico();
143
144     strcpy(mot1.laChaine, "bonjour");
145     mot1.longueur = 7;
146     strcpy(mot2.laChaine, "bonsoir");
147     mot2.longueur = 7;
148     strcpy(mot3.laChaine, "yoyo");
149     mot3.longueur = 4;
150     strcpy(mot4.laChaine, "test");
151     mot4.longueur = 4;
152     strcpy(mot5.laChaine, "écrire");
153     mot5.longueur = 6;
154
155     fils = D_chargerDico("dicoTest.txt");
156     AbN_fixerFils(&dico, fils);
157
158    CU_ASSERT_TRUE(D_estPresent(fils, mot1));
159    CU_ASSERT_TRUE(D_estPresent(fils, mot2));
160    CU_ASSERT_TRUE(D_estPresent(fils, mot3));
161    CU_ASSERT_TRUE(D_estPresent(fils, mot4));
162    CU_ASSERT_TRUE(D_estPresent(fils, mot5));
163
164     free(fils);
165     free(dico);
166 }
167
168 void test_D_sauvegarderDico(void) {
169    D_Dictionnaire dico, fils;
170    M_Mot mot1, mot2, mot3, mot4, mot5, mot6, mot7;
171
172     dico = D_creerDico();
173     fils = AbN_obtenirFils(dico);
174     strcpy(mot1.laChaine, "bonjour");
175     mot1.longueur = 7;
176     strcpy(mot2.laChaine, "bonsoir");
177     mot2.longueur = 7;
178     strcpy(mot3.laChaine, "yoyo");
179     mot3.longueur = 4;
180     strcpy(mot4.laChaine, "test");
181     mot4.longueur = 4;
182     strcpy(mot5.laChaine, "rejouer");
```

```
183     mot5.longueur = 7;
184     strcpy(mot6.laChaine, "bon");
185     mot6.longueur = 3;
186     strcpy(mot7.laChaine, "écrire");
187     mot7.longueur = 6;
188 
189     D_insererMot(&fils, mot1);
190     D_insererMot(&fils, mot3);
191     D_insererMot(&fils, mot2);
192     D_insererMot(&fils, mot6);
193     D_insererMot(&fils, mot5);
194     D_insererMot(&fils, mot7);
195     D_insererMot(&fils, mot4);
196 
197     AbN_fixerFils(&dico, fils);
198 
199     D_sauvegarderDico("dicoTest.txt", fils);
200 
201     free(fils);
202     free(dico);
203 
204     dico = D_creerDico();
205     fils = D_chargerDico("dicoTest.txt");
206     AbN_fixerFils(&dico, fils);
207 
208     CU_ASSERT_TRUE(D_estPresent(fils, mot1));
209     CU_ASSERT_TRUE(D_estPresent(fils, mot2));
210     CU_ASSERT_TRUE(D_estPresent(fils, mot3));
211     CU_ASSERT_TRUE(D_estPresent(fils, mot6));
212     CU_ASSERT_TRUE(D_estPresent(fils, mot4));
213     CU_ASSERT_TRUE(D_estPresent(fils, mot7));
214 
215     free(fils);
216     free(dico);
217 }
218 
219 int main(int argc, char** argv){
220     CU_pSuite pSuite = NULL;
221 
222 
223     if (CUE_SUCCESS != CU_initialize_registry())
224         return CU_get_error();
```

```
225
226
227     pSuite = CU_add_suite("Tests boite noire : dictionnaireTU.c",
228                           init_suite_success, clean_suite_success);
229     if (NULL == pSuite) {
230       CU_cleanup_registry();
231       return CU_get_error();
232     }
233
234
235     if ((NULL == CU_add_test(pSuite, "D_estVide", test_D_estVide))
236         || (NULL == CU_add_test(pSuite, "D_sauvegarderDico",
237                                 test_D_sauvegarderDico))
238         || (NULL == CU_add_test(pSuite, "D_insererMot", test_D_insererMot))
239         || (NULL == CU_add_test(pSuite, "D_chargerDico", test_D_chargerDico))
240         || (NULL == CU_add_test(pSuite, "D_estPresent", test_D_estPresent))
241         )
242       {
243         CU_cleanup_registry();
244         return CU_get_error();
245       }
246
247
248     CU_basic_set_mode(CU_BRM_VERBOSE);
249     CU_basic_run_tests();
250     printf("\n");
251     CU_basic_show_failures(CU_get_failure_list());
252     printf("\n\n");
253
254
255     CU_cleanup_registry();
256     return CU_get_error();
257 }
```

9.3 correcteurOrthographiqueTU.c

```
1 #include <stdlib.h>
2 #include <CUnit/Basic.h>
3 #include <string.h>
4 #include "mot.h"
5 #include "ensemble.h"
6 #include "dictionnaire.h"
```

```
7  #include "correcteurOrthographique.h"
8
9  #define TRUE 1
10 #define FALSE 0
11
12 int init_suite_success(void) {
13   return 0;
14 }
15
16 int clean_suite_success(void) {
17   return 0;
18 }
19
20 void test_CO_obtenirMot(void){
21   M_Mot lemot;
22   CO_CorrecteurOrthographique corr;
23
24   strcpy(lemot.laChaine, "bonjour");
25   lemot.longueur = 7;
26   corr=CO_correcteurOrthographique();
27   corr.mot=lemot;
28
29   CU_ASSERT_TRUE(M_sontEgaux(lemot,CO_obtenirMot(corr)));
30
31   strcpy(lemot.laChaine, "écrire");
32   lemot.longueur = 6;
33   corr=CO_correcteurOrthographique();
34   corr.mot=lemot;
35
36   CU_ASSERT_TRUE(M_sontEgaux(lemot,CO_obtenirMot(corr)));
37 }
38
39 void test_CO_fixerMot(void){
40   M_Mot lemot;
41   CO_CorrecteurOrthographique corr;
42
43   strcpy(lemot.laChaine, "bonjour");
44   lemot.longueur = 7;
45   corr=CO_correcteurOrthographique();
46   CO_fixerMot(&corr,lemot);
47
48   CU_ASSERT_TRUE(M_sontEgaux(lemot,corr.mot));
```

```
49
50    strcpy(lemot.laChaine, "écrire");
51    lemot.longueur = 7;
52    corr=CO_correcteurOrthographique();
53    CO_fixerMot(&corr,lemot);
54
55    CU_ASSERT_TRUE(M_sontEgaux(lemot,corr.mot));
56  }
57
58  void test_CO_fixerDictionnaire(void){
59    D_Dictionnaire ledico, fils;
60    CO_CorrecteurOrthographique corr;
61    M_Mot mot1, mot2, mot3, mot4, mot5;
62    ledico=D_creerDico();
63    fils=AbN_obtenirFils(ledico);
64    corr=CO_correcteurOrthographique();
65
66    strcpy(mot1.laChaine, "bonjour");
67    mot1.longueur = 7;
68    strcpy(mot2.laChaine, "bonsoir");
69    mot2.longueur = 7;
70    strcpy(mot3.laChaine, "yoyo");
71    mot3.longueur = 4;
72    strcpy(mot4.laChaine, "test");
73    mot4.longueur = 4;
74    strcpy(mot5.laChaine, "écrire");
75    mot5.longueur = 6;
76
77    D_insererMot(&fils, mot1);
78    D_insererMot(&fils, mot2);
79    D_insererMot(&fils, mot3);
80    D_insererMot(&fils, mot4);
81    D_insererMot(&fils, mot5);
82
83    AbN_fixerFils(&ledico, fils);
84
85    CO_fixerDictionnaire(&corr,ledico);
86
87    CU_ASSERT_TRUE(ledico==corr.dico);
88
89    free(fils);
90    free(ledico);
```

```
91  }
92
93  void test_CO_obtenirDictionnaire(void){
94    D_Dictionnaire ledico, fils;
95    CO_CorrecteurOrthographique corr;
96    M_Mot mot1, mot2, mot3, mot4, mot5;
97    ledico=D_creerDico();
98    fils=AbN_obtenirFils(ledico);
99    corr=CO_correcteurOrthographique();
100
101   strcpy(mot1.laChaine, "bonjour");
102   mot1.longueur = 7;
103   strcpy(mot2.laChaine, "bonsoir");
104   mot2.longueur = 7;
105   strcpy(mot3.laChaine, "yoyo");
106   mot3.longueur = 4;
107   strcpy(mot4.laChaine, "test");
108   mot4.longueur = 4;
109   strcpy(mot5.laChaine, "écrire");
110   mot5.longueur = 6;
111
112   D_insererMot(&fils, mot1);
113   D_insererMot(&fils, mot2);
114   D_insererMot(&fils, mot3);
115   D_insererMot(&fils, mot4);
116   D_insererMot(&fils, mot5);
117
118   AbN_fixerFils(&ledico, fils);
119
120   corr.dico=ledico;
121
122   CU_ASSERT_TRUE(ledico==CO_obtenirDictionnaire(corr));
123
124   free(fils);
125   free(ledico);
126 }
127
128 void test_CO_estBienOrthographie(void){
129   D_Dictionnaire ledico, fils;
130   CO_CorrecteurOrthographique corr;
131   M_Mot mot1, mot2, mot3, mot4, mot5, motf1, motf2;
132   ledico=D_creerDico();
```

```
133     fils=AbN_obtenirFils(ledico);
134     corr=CO_correcteurOrthographique();
135
136     strcpy(mot1.laChaine, "bonjour");
137     mot1.longueur = 7;
138     strcpy(mot2.laChaine, "bonsoir");
139     mot2.longueur = 7;
140     strcpy(mot3.laChaine, "yoyo");
141     mot3.longueur = 4;
142     strcpy(mot4.laChaine, "test");
143     mot4.longueur = 4;
144     strcpy(mot5.laChaine, "écrire");
145     mot5.longueur = 6;
146
147     D_insererMot(&fils, mot1);
148     D_insererMot(&fils, mot2);
149     D_insererMot(&fils, mot5);
150     D_insererMot(&fils, mot3);
151     D_insererMot(&fils, mot4);
152
153     AbN_fixerFils(&ledico, fils);
154
155     CO_fixerDictionnaire(&corr,fils);
156     CO_fixerMot(&corr,mot1);
157     CU_ASSERT_TRUE(CO_estBienOrthographie(corr));
158
159     CO_fixerMot(&corr,mot5);
160     CU_ASSERT_TRUE(CO_estBienOrthographie(corr));
161
162     strcpy(motf1.laChaine, "bonjoir");
163     motf1.longueur = 7;
164     CO_fixerMot(&corr,motf1);
165     CU_ASSERT_TRUE(CO_estBienOrthographie(corr)==FALSE);
166
167     strcpy(motf2.laChaine, "écrir");
168     motf2.longueur = 5;
169     CO_fixerMot(&corr,motf2);
170     CU_ASSERT_TRUE(CO_estBienOrthographie(corr)==FALSE);
171
172     free(fils);
173     free(ledico);
174 }
```

```
175
176 void test_CO_proposerCorrections (void) {
177   D_Dictionnaire ledico, fils;
178   E_Ensemble lensemble;
179   CO_CorrecteurOrthographique corr;
180   M_Mot mot1, mot2, mot3, mot4, mot5, mot6, mot7, mot8, mot9, mot10, mot11,
181     mot12;
182   M_Mot motf1, motf2, motf3, motf4, motf5, motf6;
183   lensemble=E_ensemble();
184   ledico=D_creerDico();
185   fils=AbN_obtenirFils(ledico);
186   corr=CO_correcteurOrthographique();
187
188   strcpy(mot1.laChaine, "bonjour");
189   mot1.longueur = 7;
190   strcpy(mot2.laChaine, "bonsoir");
191   mot2.longueur = 7;
192   strcpy(mot3.laChaine, "yoyo");
193   mot3.longueur = 4;
194   strcpy(mot4.laChaine, "test");
195   mot4.longueur = 4;
196   strcpy(mot5.laChaine, "yin");
197   mot5.longueur = 3;
198   strcpy(mot6.laChaine, "yen");
199   mot6.longueur = 3;
200   strcpy(mot7.laChaine, "un");
201   mot7.longueur = 2;
202   strcpy(mot8.laChaine, "on");
203   mot8.longueur = 2;
204   strcpy(mot9.laChaine, "in");
205   mot9.longueur = 2;
206
207   strcpy(mot10.laChaine, "en");
208   mot10.longueur = 2;
209   strcpy(mot11.laChaine, "an");
210   mot11.longueur = 2;
211   strcpy(mot12.laChaine, "écrire");
212   mot12.longueur = 6;
213
214   D_insererMot(&fils, mot1);
215   D_insererMot(&fils, mot2);
216   D_insererMot(&fils, mot3);
```

```
217     D_insererMot(&fils , mot4);
218     D_insererMot(&fils , mot5);
219     D_insererMot(&fils , mot6);
220     D_insererMot(&fils , mot7);
221     D_insererMot(&fils , mot12);
222     D_insererMot(&fils , mot8);
223     D_insererMot(&fils , mot9);
224
225     AbN_fixerFils(&ledico , fils);
226
227     CO_fixerDictionnaire(&corr ,fils);
228
229     strcpy(motf1.laChaine , "bonjoir");
230     motf1.longueur = 7;
231     strcpy(motf2.laChaine , "bonsoi");
232     motf2.longueur = 6;
233     strcpy(motf3.laChaine , "yoyos");
234     motf3.longueur = 5;
235     strcpy(motf4.laChaine , "tset");
236     motf4.longueur = 4;
237     strcpy(motf5.laChaine , "yn");
238     motf5.longueur = 2;
239     strcpy(motf6.laChaine , "écire");
240     motf6.longueur = 5;
241
242     CO_fixerMot(&corr ,motf1);
243     lensemble=CO_proposerCorrections(corr);
244     CU_ASSERT_TRUE(E_estPresent(lensemble ,mot1));
245
246     CO_fixerMot(&corr ,motf2);
247     lensemble=CO_proposerCorrections(corr);
248     CU_ASSERT_TRUE(E_estPresent(lensemble ,mot2));
249
250     CO_fixerMot(&corr ,motf3);
251     lensemble=CO_proposerCorrections(corr);
252     CU_ASSERT_TRUE(E_estPresent(lensemble ,mot3));
253
254     CO_fixerMot(&corr ,motf4);
255     lensemble=CO_proposerCorrections(corr);
256     CU_ASSERT_TRUE(E_estPresent(lensemble ,mot4));
257
258     CO_fixerMot(&corr ,motf6);
```

```
259     lensemble=CO_proposerCorrections(corr);
260     CU_ASSERT_TRUE(E_estPresent(lensemble,mot12));
261
262
263     CO_fixerMot(&corr,motf5);
264     lensemble=CO_proposerCorrections(corr);
265     CU_ASSERT_TRUE(E_estPresent(lensemble,mot5)==TRUE);
266
267     CU_ASSERT_TRUE(E_estPresent(lensemble,mot6)==TRUE);
268     CU_ASSERT_TRUE(E_estPresent(lensemble,mot7)==TRUE);
269     CU_ASSERT_TRUE(E_estPresent(lensemble,mot8)==TRUE);
270     CU_ASSERT_TRUE(E_estPresent(lensemble,mot9)==TRUE);
271     CU_ASSERT_TRUE(E_estPresent(lensemble,mot10)==FALSE);
272     CU_ASSERT_TRUE(E_estPresent(lensemble,mot11)==FALSE);
273
274
275     CU_ASSERT_TRUE(E_estPresent(lensemble,mot12)==FALSE);
276
277     free(fils);
278     free(ledico);
279 }
280
281 int main(int argc, char** argv){
282
283     CU_pSuite pSuite = NULL;
284
285
286     if (CUE_SUCCESS != CU_initialize_registry())
287       return CU_get_error();
288
289
290     pSuite = CU_add_suite("Tests boite noire : correcteurOrthographiqueTU.c",
291                           init_suite_success, clean_suite_success);
292     if (NULL == pSuite) {
293       CU_cleanup_registry();
294       return CU_get_error();
295     }
296
297
298     if ((NULL == CU_add_test(pSuite, "CO_obtenirMot", test_CO_obtenirMot))
299         ||(NULL == CU_add_test(pSuite, "CO_obtenirDictionnaire",
300                                test_CO_obtenirDictionnaire))
```

```
301         ||(NULL == CU_add_test(pSuite, "CO_fixerDictionnaire",
302                                    test_CO_fixerDictionnaire))
303         ||(NULL == CU_add_test(pSuite, "CO_fixerMot", test_CO_fixerMot))
304         ||(NULL == CU_add_test(pSuite, "CO_estBienOrthographie",
305                                    test_CO_estBienOrthographie))
306         ||(NULL == CU_add_test(pSuite, "CO_proposerCorrections",
307                                    test_CO_proposerCorrections))
308         )
309       {
310         CU_cleanup_registry();
311         return CU_get_error();
312       }
313
314
315     CU_basic_set_mode(CU_BRM_VERBOSE);
316     CU_basic_run_tests();
317     printf("\n");
318     CU_basic_show_failures(CU_get_failure_list());
319     printf("\n\n");
320
321
322     CU_cleanup_registry();
323     return CU_get_error();
324 }
```

9.4 arbreNTU.c

```
1  #include <stdlib.h>
2  #include <CUnit/Basic.h>
3  #include <string.h>
4  #include "arbreN.h"
5
6  #define TRUE 1
7  #define FALSE 0
8  #define charnul '\0'
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_AbN_creerArbreNonInit(void) {
19   AbN_ArbreN a;
20   a = AbN_creerArbreNonInit();
21
22   CU_ASSERT_TRUE((*a).lettre == charnul);
23   CU_ASSERT_TRUE((*a).motValide == FALSE);
24   CU_ASSERT_TRUE((*a).Fils == NULL);
25   CU_ASSERT_TRUE((*a).Frere == NULL);
26 }
27
28 void test_AbN_estVide(void) {
29   AbN_ArbreN a;
30   a = NULL;
31
32   CU_ASSERT_TRUE(AbN_estVide(a));
33 }
34
35 void test_AbN_obtenirBool(void) {
36   AbN_ArbreN a;
37
38   a = AbN_creerArbreNonInit();
39   CU_ASSERT_TRUE(AbN_obtenirBool(a) == FALSE);
40
41   a->motValide = TRUE;
42   CU_ASSERT_TRUE(AbN_obtenirBool(a) == TRUE);
43 }
44
45 void test_AbN_obtenirChar(void) {
46   AbN_ArbreN a;
47
48   a = AbN_creerArbreNonInit();
49   CU_ASSERT_TRUE(AbN_obtenirChar(a) == charnul);
50
51   a->lettre = 'b';
52   CU_ASSERT_TRUE(AbN_obtenirChar(a) == 'b');
53
54   a = AbN_creerArbreNonInit();
55   CU_ASSERT_TRUE(AbN_obtenirChar(a) == charnul);
56
57   a->lettre = 'é';
```

```
58    CU_ASSERT_TRUE(AbN_obtenirChar(a) == 'é');
59  }
60
61  void test_AbN_fixerBool(void) {
62    AbN_ArbreN a;
63
64    a = AbN_creerArbreNonInit();
65    AbN_fixerBool(&a, TRUE);
66    CU_ASSERT_TRUE(AbN_obtenirBool(a) == TRUE);
67
68    AbN_fixerBool(&a, FALSE);
69    CU_ASSERT_TRUE(AbN_obtenirBool(a) == FALSE);
70  }
71
72  void test_AbN_fixerChar(void) {
73    AbN_ArbreN a;
74
75    a = AbN_creerArbreNonInit();
76    AbN_fixerChar(&a, 'b');
77    CU_ASSERT_TRUE(AbN_obtenirChar(a) == 'b');
78
79    AbN_fixerChar(&a, 'é');
80    CU_ASSERT_TRUE(AbN_obtenirChar(a) == 'é');
81
82    AbN_fixerChar(&a, charnul);
83    CU_ASSERT_TRUE(AbN_obtenirChar(a) == charnul);
84  }
85
86  void test_AbN_obtenirFrere(void) {
87    AbN_ArbreN a, a1, a2, a3;
88
89    a = AbN_creerArbreNonInit();
90    CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFrere(a)));
91
92    a1 = AbN_creerArbreNonInit();
93    AbN_fixerBool(&a1, TRUE);
94    AbN_fixerChar(&a1, 'b');
95    AbN_fixerFrere(&a, a1);
96
97    CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFrere(a))));
98    CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFrere(a)) == 'b');
99    CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFrere(a)) == TRUE);
```

```
100
101     a2 = AbN_creerArbreNonInit();
102     CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFrere(a2)));
103
104     a3 = AbN_creerArbreNonInit();
105     AbN_fixerBool(&a3, FALSE);
106     AbN_fixerChar(&a3, 'é');
107     AbN_fixerFrere(&a2, a3);
108
109     CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFrere(a2))));
110     CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFrere(a2)) == 'é');
111     CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFrere(a2)) == FALSE);
112 }
113
114 void test_AbN_obtenirFils(void) {
115     AbN_ArbreN a, a1, a2, a3;
116
117     a = AbN_creerArbreNonInit();
118     CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFils(a)));
119
120     a1 = AbN_creerArbreNonInit();
121     AbN_fixerBool(&a1, TRUE);
122     AbN_fixerChar(&a1, 'b');
123     AbN_fixerFils(&a, a1);
124
125     CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFils(a))));
126     CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFils(a)) == 'b');
127     CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFils(a)) == TRUE);
128
129     a2 = AbN_creerArbreNonInit();
130     CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFils(a2)));
131
132     a3 = AbN_creerArbreNonInit();
133     AbN_fixerBool(&a3, FALSE);
134     AbN_fixerChar(&a3, 'é');
135     AbN_fixerFils(&a2, a3);
136
137     CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFils(a2))));
138     CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFils(a2)) == 'é');
139     CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFils(a2)) == FALSE);
140 }
141
```

```
142 void test_AbN_fixerFrere(void) {
143   AbN_ArbreN a, a1, a2, a3;
144
145   a = AbN_creerArbreNonInit();
146   CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFrere(a)));
147
148   a1 = AbN_creerArbreNonInit();
149   AbN_fixerBool(&a1, TRUE);
150   AbN_fixerChar(&a1, 'b');
151   AbN_fixerFrere(&a, a1);
152
153   CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFrere(a))));
154   CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFrere(a)) == 'b');
155   CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFrere(a)) == TRUE);
156
157   a2 = AbN_creerArbreNonInit();
158   CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFrere(a2)));
159
160   a3 = AbN_creerArbreNonInit();
161   AbN_fixerBool(&a3, FALSE);
162   AbN_fixerChar(&a3, 'é');
163   AbN_fixerFrere(&a2, a3);
164
165   CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFrere(a2))));
166   CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFrere(a2)) == 'é');
167   CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFrere(a2)) == FALSE);
168 }
169
170 void test_AbN_fixerFils(void) {
171   AbN_ArbreN a, a1, a2, a3;
172
173   a = AbN_creerArbreNonInit();
174   CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFils(a)));
175
176   a1 = AbN_creerArbreNonInit();
177   AbN_fixerBool(&a1, TRUE);
178   AbN_fixerChar(&a1, 'b');
179   AbN_fixerFils(&a, a1);
180
181   CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFils(a))));
182   CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFils(a)) == 'b');
183   CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFils(a)) == TRUE);
```

```
184
185   a2 = AbN_creerArbreNonInit();
186   CU_ASSERT_TRUE(AbN_estVide(AbN_obtenirFils(a2)));
187
188   a3 = AbN_creerArbreNonInit();
189   AbN_fixerBool(&a3, FALSE);
190   AbN_fixerChar(&a3, 'é');
191   AbN_fixerFils(&a2, a3);
192
193   CU_ASSERT_TRUE(!(AbN_estVide(AbN_obtenirFils(a2))));
194   CU_ASSERT_TRUE(AbN_obtenirChar(AbN_obtenirFils(a2)) == 'é');
195   CU_ASSERT_TRUE(AbN_obtenirBool(AbN_obtenirFils(a2)) == FALSE);
196 }
197
198
199 int main(int argc, char** argv){
200   CU_pSuite pSuite = NULL;
201
202
203   if (CUE_SUCCESS != CU_initialize_registry())
204     return CU_get_error();
205
206
207   pSuite = CU_add_suite("Tests boite noire : arbreNTU.c",
208                         init_suite_success, clean_suite_success);
209   if (NULL == pSuite) {
210     CU_cleanup_registry();
211     return CU_get_error();
212   }
213
214
215   if ((NULL == CU_add_test(pSuite, "AbN_creerArbreNonInit",
216                            test_AbN_creerArbreNonInit))
217       || (NULL == CU_add_test(pSuite, "AbN_estVide", test_AbN_estVide))
218       || (NULL == CU_add_test(pSuite, "AbN_obtenirBool", test_AbN_obtenirBool))
219       || (NULL == CU_add_test(pSuite, "AbN_obtenirChar", test_AbN_obtenirChar))
220       || (NULL == CU_add_test(pSuite, "AbN_fixerBool", test_AbN_fixerBool))
221       || (NULL == CU_add_test(pSuite, "AbN_fixerChar", test_AbN_fixerChar))
222       || (NULL == CU_add_test(pSuite, "AbN_obtenirFils", test_AbN_obtenirFils))
223       || (NULL == CU_add_test(pSuite, "AbN_obtenirFrere",
224                               test_AbN_obtenirFrere))
225       || (NULL == CU_add_test(pSuite, "AbN_obtenirFils", test_AbN_fixerFils))
```

```
226         || (NULL == CU_add_test(pSuite, "AbN_obtenirFrere", test_AbN_fixerFrere))
227         )
228       {
229         CU_cleanup_registry();
230         return CU_get_error();
231       }
232
233
234     CU_basic_set_mode(CU_BRM_VERBOSE);
235     CU_basic_run_tests();
236     printf("\n");
237     CU_basic_show_failures(CU_get_failure_list());
238     printf("\n\n");
239
240
241     CU_cleanup_registry();
242     return CU_get_error();
243 }
```

9.5 ensembleTU.c

```
1  #include <stdlib.h>
2  #include <CUnit/Basic.h>
3  #include <string.h>
4  #include "mot.h"
5  #include "ensemble.h"
6  #include "listeChaineeMot.h"
7
8  #define TRUE 1
9  #define FALSE 0
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_E_ensemble(void) {
20   E_Ensemble e;
21   e=E_ensemble();
```

```
22
23    CU_ASSERT_TRUE(e.lesElements==LCM_listeChaineeMot());
24    CU_ASSERT_TRUE(e.nbElement==0);
25  }
26
27  void test_E_obtenirLesElements(void) {
28    E_Ensemble e;
29    M_Mot mot;
30    E_Element el;
31
32    e=E_ensemble();
33    M_fixerLaChaine(&mot, "bonsoir");
34    M_fixerLongueur(&mot, 7);
35    el=mot;
36    E_ajouter(&e, el);
37
38    CU_ASSERT_TRUE(!(LCM_estVide(E_obtenirLesElements(e))));
39
40    e=E_ensemble();
41    M_fixerLaChaine(&mot, "écrire");
42    M_fixerLongueur(&mot, 6);
43    el=mot;
44    E_ajouter(&e, el);
45
46    CU_ASSERT_TRUE(!(LCM_estVide(E_obtenirLesElements(e))));
47  }
48
49  void test_E_obtenirNbElement(void) {
50    E_Ensemble e;
51
52    e=E_ensemble();
53    e.nbElement=3;
54
55    CU_ASSERT_TRUE(E_obtenirNbElement(e)==3);
56  }
57
58  void test_E_cardinalite(void) {
59    E_Ensemble e;
60
61    e=E_ensemble();
62    e.nbElement=3;
63
```

```
64    CU_ASSERT_TRUE(E_cardinalite(e)==3);
65  }
66
67  void test_E_fixerLesElements(void) {
68    E_Ensemble e1, e2;
69    M_Mot mot;
70    E_Element el;
71
72    e1=E_ensemble();
73    M_fixerLaChaine(&mot, "bonsoir");
74    M_fixerLongueur(&mot, 7);
75    el=mot;
76    E_ajouter(&e1, el);
77
78    e2=E_ensemble();
79    M_fixerLaChaine(&mot, "hello");
80    M_fixerLongueur(&mot, 5);
81    el=mot;
82    E_ajouter(&e2, el);
83
84    E_fixerLesElements(&e1, e2.lesElements);
85
86    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(
87                LCM_obtenirListeSuivante(E_obtenirLesElements(e1)))), "hello")==0);
88    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(LCM_obtenirListeSuivante(
89                                                E_obtenirLesElements(e1))))==5);
90
91    e1=E_ensemble();
92    M_fixerLaChaine(&mot, "bonsoir");
93    M_fixerLongueur(&mot, 7);
94    el=mot;
95    E_ajouter(&e1, el);
96
97    e2=E_ensemble();
98    M_fixerLaChaine(&mot, "écrire");
99    M_fixerLongueur(&mot, 6);
100   el=mot;
101   E_ajouter(&e2, el);
102
103   E_fixerLesElements(&e1, e2.lesElements);
104
105   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(
```

```
106                 LCM_obtenirListeSuivante(E_obtenirLesElements(e1)))), "écrire")==0);
107     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(LCM_obtenirListeSuivante(
108                                                         E_obtenirLesElements(e1))))==6);
109 }
110
111 void test_E_ajouter(void) {
112     E_Ensemble e;
113     M_Mot mot;
114     LCM_ListeChaineeMot listeTest;
115
116     e=E_ensemble();
117
118     M_fixerLaChaine(&mot, "bonjour");
119     M_fixerLongueur(&mot, 7);
120
121     listeTest=LCM_listeChaineeMot();
122
123     E_ajouter(&e, mot);
124
125     listeTest=E_obtenirLesElements(e);
126
127     CU_ASSERT_TRUE(!(LCM_estVide(listeTest)));
128     CU_ASSERT_TRUE(E_obtenirNbElement(e)==1);
129
130     e=E_ensemble();
131
132     M_fixerLaChaine(&mot, "écrire");
133     M_fixerLongueur(&mot, 6);
134
135     listeTest=LCM_listeChaineeMot();
136
137     E_ajouter(&e, mot);
138
139     listeTest=E_obtenirLesElements(e);
140
141     CU_ASSERT_TRUE(!(LCM_estVide(listeTest)));
142     CU_ASSERT_TRUE(E_obtenirNbElement(e)==1);
143 }
144
145 void test_E_estPresent(void) {
146     E_Ensemble e;
147     M_Mot mot1, mot2;
```

```
148     E_Element el1, el2;
149
150     e=E_ensemble();
151
152     M_fixerLaChaine(&mot1, "bonjour");
153     M_fixerLongueur(&mot1, 7);
154     el1=mot1;
155
156     M_fixerLaChaine(&mot2, "bonsoir");
157     M_fixerLongueur(&mot2, 7);
158     el2=mot2;
159
160     E_ajouter(&e, el1);
161     E_ajouter(&e, el2);
162
163     CU_ASSERT_TRUE(E_estPresent(e, el1));
164     CU_ASSERT_TRUE(E_estPresent(e, el2));
165
166     e=E_ensemble();
167
168     M_fixerLaChaine(&mot1, "écrire");
169     M_fixerLongueur(&mot1, 6);
170     el1=mot1;
171
172     M_fixerLaChaine(&mot2, "bonsoir");
173     M_fixerLongueur(&mot2, 7);
174     el2=mot2;
175
176     E_ajouter(&e, el1);
177     E_ajouter(&e, el2);
178
179     CU_ASSERT_TRUE(E_estPresent(e, el1));
180     CU_ASSERT_TRUE(E_estPresent(e, el2));
181 }
182
183 void test_E_retirer(void) {
184     E_Ensemble e;
185     M_Mot mot1, mot2, mot3, mot4;
186     E_Element el1, el2, el3, el4;
187
188     e=E_ensemble();
189
```

```
190     M_fixerLaChaine(&mot1, "bonjour");
191     M_fixerLongueur(&mot1, 7);
192     el1=mot1;
193
194     M_fixerLaChaine(&mot2, "bonsoir");
195     M_fixerLongueur(&mot2, 7);
196     el2=mot2;
197
198     M_fixerLaChaine(&mot3, "hello");
199     M_fixerLongueur(&mot3, 5);
200     el3=mot3;
201
202     M_fixerLaChaine(&mot4, "écrire");
203     M_fixerLongueur(&mot4, 6);
204     el4=mot4;
205
206     E_ajouter(&e, el1);
207     E_ajouter(&e, el2);
208     E_ajouter(&e, el4);
209
210     CU_ASSERT_TRUE(E_estPresent(e, el3)==FALSE);
211     E_retirer(&e, el3);
212     CU_ASSERT_TRUE(E_cardinalite(e)==3);
213
214     E_retirer(&e, el1);
215
216     CU_ASSERT_TRUE(E_estPresent(e, el1)==FALSE);
217     CU_ASSERT_TRUE(E_estPresent(e, el2)==TRUE);
218     CU_ASSERT_TRUE(E_estPresent(e, el4)==TRUE);
219     CU_ASSERT_TRUE(E_cardinalite(e)==2);
220
221     E_retirer(&e, el2);
222
223     CU_ASSERT_TRUE(E_estPresent(e, el2)==FALSE);
224     CU_ASSERT_TRUE(E_cardinalite(e)==1);
225
226     E_retirer(&e, el4);
227     CU_ASSERT_TRUE(E_estPresent(e, el4)==FALSE);
228     CU_ASSERT_TRUE(E_cardinalite(e)==0);
229 }
230
231 int main(int argc, char** argv){
```

```
232   CU_pSuite pSuite = NULL;
233
234
235   if (CUE_SUCCESS != CU_initialize_registry())
236     return CU_get_error();
237
238
239   pSuite = CU_add_suite("Tests boite noire : ensembleTU.c",
240                         init_suite_success, clean_suite_success);
241   if (NULL == pSuite) {
242     CU_cleanup_registry();
243     return CU_get_error();
244   }
245
246
247   if ((NULL == CU_add_test(pSuite, "E_ensemble", test_E_ensemble))
248       || (NULL == CU_add_test(pSuite, "E_obtenirLesElements",
249                               test_E_obtenirLesElements))
250       || (NULL == CU_add_test(pSuite, "E_obtenirNbElement",
251                               test_E_obtenirNbElement))
252       || (NULL == CU_add_test(pSuite, "E_fixerLesElements",
253                               test_E_fixerLesElements))
254       || (NULL == CU_add_test(pSuite, "E_ajouter", test_E_ajouter))
255       || (NULL == CU_add_test(pSuite, "E_cardinalite", test_E_cardinalite))
256       || (NULL == CU_add_test(pSuite, "E_estPresent", test_E_estPresent))
257       || (NULL == CU_add_test(pSuite, "E_retirer", test_E_retirer))
258       )
259     {
260       CU_cleanup_registry();
261       return CU_get_error();
262     }
263
264
265   CU_basic_set_mode(CU_BRM_VERBOSE);
266   CU_basic_run_tests();
267   printf("\n");
268   CU_basic_show_failures(CU_get_failure_list());
269   printf("\n\n");
270
271
272   CU_cleanup_registry();
273   return CU_get_error();
```

```
274 }
```

9.6 listeChaineeMotTU.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <CUnit/Basic.h>
4  #include <string.h>
5  #include "listeChaineeMot.h"
6  #include "mot.h"
7
8  #define TRUE 1
9  #define FALSE 0
10
11 int init_suite_success(void) {
12   return 0;
13 }
14
15 int clean_suite_success(void) {
16   return 0;
17 }
18
19
20 void initialiserListePourTestsUnitaires(LCM_ListeChaineeMot *l, char *laChaine,
21                                         unsigned int longueur) {
22   M_Mot mot;
23
24   M_fixerLaChaine(&mot, laChaine);
25   M_fixerLongueur(&mot, longueur);
26   LCM_ajouter(l, mot);
27 }
28
29
30 void test_LCM_listeChaineeMot(void) {
31   CU_ASSERT_TRUE(LCM_listeChaineeMot()==NULL);
32 }
33
34 void test_LCM_estVide(void) {
35   LCM_ListeChaineeMot l;
36
37   l=LCM_listeChaineeMot();
38   CU_ASSERT_TRUE(LCM_estVide(l)==TRUE);
```

```
39
40    initialiserListePourTestsUnitaires(&l, "bonsoir", 7);
41    CU_ASSERT_TRUE(LCM_estVide(l)==FALSE);
42
43    l=LCM_listeChaineeMot();
44    CU_ASSERT_TRUE(LCM_estVide(l)==TRUE);
45
46    initialiserListePourTestsUnitaires(&l, "écrire", 6);
47    CU_ASSERT_TRUE(LCM_estVide(l)==FALSE);
48  }
49
50  void test_LCM_ajouter(void) {
51    LCM_ListeChaineeMot l;
52
53    l=LCM_listeChaineeMot();
54
55    initialiserListePourTestsUnitaires(&l, "bonsoir", 7);
56    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
57                          "bonsoir")==0);
58    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==7);
59
60    initialiserListePourTestsUnitaires(&l, "hello", 5);
61    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)), "hello")==0);
62    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==5);
63
64    initialiserListePourTestsUnitaires(&l, "écrire", 6);
65    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
66                          "écrire")==0);
67    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==6);
68  }
69
70  void test_LCM_obtenirElement(void) {
71    LCM_ListeChaineeMot l;
72
73    l=LCM_listeChaineeMot();
74
75    initialiserListePourTestsUnitaires(&l, "bonsoir", 7);
76    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
77                          "bonsoir")==0);
78    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==7);
79
80    initialiserListePourTestsUnitaires(&l, "hello", 5);
```

```
81    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
82                                "hello")==0);
83    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==5);
84
85    initialiserListePourTestsUnitaires(&l, "écrire", 5);
86    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
87                                "écrire")==0);
88    CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==5);
89  }
90
91  void test_LCM_obtenirListeSuivante(void) {
92    LCM_ListeChaineeMot l;
93
94    l=LCM_listeChaineeMot();
95
96    initialiserListePourTestsUnitaires(&l, "bonsoir", 7);
97    initialiserListePourTestsUnitaires(&l, "hello", 5);
98
99    CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(
100                                    LCM_obtenirListeSuivante(l))), "bonsoir")==0);
101   CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(
102                                            LCM_obtenirListeSuivante(l)))==7);
103
104   l=LCM_listeChaineeMot();
105
106   initialiserListePourTestsUnitaires(&l, "écrire", 6);
107   initialiserListePourTestsUnitaires(&l, "hello", 5);
108
109   CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(
110                                     LCM_obtenirListeSuivante(l))), "écrire")==0);
111   CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(
112                                            LCM_obtenirListeSuivante(l)))==6);
113 }
114
115 void test_LCM_fixerListeSuivante(void) {
116   LCM_ListeChaineeMot l1, l2;
117
118   l1=LCM_listeChaineeMot();
119   l2=LCM_listeChaineeMot();
120
121   initialiserListePourTestsUnitaires(&l1, "bonsoir", 7);
122   initialiserListePourTestsUnitaires(&l2, "hello", 5);
```

```
123
124     LCM_fixerListeSuivante(l1, l2);
125     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(
126                                         LCM_obtenirListeSuivante(l1))), "hello")==0);
127     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(
128                                                 LCM_obtenirListeSuivante(l1)))==5);
129
130     l1=LCM_listeChaineeMot();
131     l2=LCM_listeChaineeMot();
132
133     initialiserListePourTestsUnitaires(&l1, "écrire", 6);
134     initialiserListePourTestsUnitaires(&l2, "hello", 5);
135
136     LCM_fixerListeSuivante(l1, l2);
137     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(
138                                         LCM_obtenirListeSuivante(l1))), "hello")==0);
139     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(
140                                                 LCM_obtenirListeSuivante(l1)))==5);
141 }
142
143 void test_LCM_fixerElement(void) {
144     LCM_ListeChaineeMot l=(LCM_ListeChaineeMot)malloc(sizeof(LCM_Noeud));
145     M_Mot mot;
146
147     M_fixerLaChaine(&mot, "bonsoir");
148     M_fixerLongueur(&mot, 7);
149
150     LCM_fixerElement(l, mot);
151
152     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
153                           "bonsoir")==0);
154     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==7);
155
156     l=(LCM_ListeChaineeMot)malloc(sizeof(LCM_Noeud));
157
158     M_fixerLaChaine(&mot, "écrire");
159     M_fixerLongueur(&mot, 6);
160
161     LCM_fixerElement(l, mot);
162
163     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
164                           "écrire")==0);
```

```
165     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==6);
166   }
167
168   void test_LCM_supprimerTete(void) {
169     LCM_ListeChaineeMot l;
170
171     l=LCM_listeChaineeMot();
172
173     initialiserListePourTestsUnitaires(&l, "bonsoir", 7);
174     initialiserListePourTestsUnitaires(&l, "hello", 5);
175
176     LCM_supprimerTete(&l);
177     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
178                           "bonsoir")==0);
179     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==7);
180
181     l=LCM_listeChaineeMot();
182
183     initialiserListePourTestsUnitaires(&l, "écrire", 6);
184     initialiserListePourTestsUnitaires(&l, "hello", 5);
185
186     LCM_supprimerTete(&l);
187     CU_ASSERT_TRUE(strcmp(M_obtenirLaChaine(LCM_obtenirElement(l)),
188                           "écrire")==0);
189     CU_ASSERT_TRUE(M_obtenirLongueur(LCM_obtenirElement(l))==6);
190   }
191
192   void test_LCM_supprimer(void) {
193     LCM_ListeChaineeMot l;
194
195     l=LCM_listeChaineeMot();
196
197     initialiserListePourTestsUnitaires(&l, "bonsoir", 7);
198     initialiserListePourTestsUnitaires(&l, "hello", 5);
199     initialiserListePourTestsUnitaires(&l, "écrire", 6);
200
201     LCM_supprimer(&l);
202
203     CU_ASSERT_TRUE(LCM_estVide(l)==TRUE);
204   }
205
206
```

```
207 int main(int argc, char** argv){
208   CU_pSuite pSuite = NULL;
209
210
211   if (CUE_SUCCESS != CU_initialize_registry())
212     return CU_get_error();
213
214
215   pSuite = CU_add_suite("Tests boite noire : listeChaineeMotTU.c",
216                         init_suite_success, clean_suite_success);
217   if (NULL == pSuite) {
218     CU_cleanup_registry();
219     return CU_get_error();
220   }
221
222
223   if ((NULL == CU_add_test(pSuite, "LCM_listeChaineeMot",
224                            test_LCM_listeChaineeMot))
225       || (NULL == CU_add_test(pSuite, "LCM_estVide", test_LCM_estVide))
226       || (NULL == CU_add_test(pSuite, "LCM_ajouter", test_LCM_ajouter))
227       || (NULL == CU_add_test(pSuite, "LCM_obtenirElement",
228                               test_LCM_obtenirElement))
229       || (NULL == CU_add_test(pSuite, "LCM_obtenirListeSuivante",
230                               test_LCM_obtenirListeSuivante))
231       || (NULL == CU_add_test(pSuite, "LCM_fixerListeSuivante",
232                               test_LCM_fixerListeSuivante))
233       || (NULL == CU_add_test(pSuite, "LCM_fixerElement",
234                               test_LCM_fixerElement))
235       || (NULL == CU_add_test(pSuite, "LCM_supprimerTete",
236                               test_LCM_supprimerTete))
237       || (NULL == CU_add_test(pSuite, "LCM_supprimer", test_LCM_supprimer))
238       )
239     {
240       CU_cleanup_registry();
241       return CU_get_error();
242     }
243
244
245   CU_basic_set_mode(CU_BRM_VERBOSE);
246   CU_basic_run_tests();
247   printf("\n");
248   CU_basic_show_failures(CU_get_failure_list());
```

```
249     printf("\n\n");
250
251
252     CU_cleanup_registry();
253     return CU_get_error();
254 }
```

10 Conclusion

10.1 Conclusion générale

Ce projet était très intéressant. Le sujet du correcteur orthographique est concret et permet de voir des aspects particuliers d'un projet comme par exemple l'encodage des caractères. Il nous a ainsi permis d'apprendre le langage C, ainsi que le travail en équipe sur un problème informatique et dans des conditions réelles de projet en entreprise.

Nous sommes tout de même déçu car notre programme ne fonctionne qu'avec un dictionnaire sans accents (48843 mots) que nous avons trouvé sur internet (répertoire DicoSansAccents). Nous ne sommes pas parvenus à trouver le problème qui survient lorsque nous voulons construire notre SDD dictionnaire avec les mots du dictionnaire donné sur Moodle. Nous avons cherché et il nous semble que notre code est bon mais que nous avons des problèmes avec l'encodage latin-1.

Nous avons tenté de trouver le problème avec Valgrind et ddd mais nous n'y sommes pas parvenus. Lorsqu'on utilise le dictionnaire avec accents, en mettant la commande suivante : valgrind --leak-check=full --track-origins=yes ./bin/asispell -d ./bin/francais.dico -f ./DicoAvecAccents/dico-ref-ascii.txt, Valgrind dit entre autres : Uninitialised value was created by a stack allocation at 0x10B1F5 : D_insererMot (dictionnaire.c :130). Nous pensons donc que l'erreur survient dans D_insererMot mais pas avec plus de précision malheureusement.

10.2 Conclusions personnelles

10.2.1 Dounia BOUTAYEB

Ce projet s'est révélé très enrichissant dans la mesure où il a consisté en une approche concrète du métier d'ingénieur. En effet, la prise d'initiative, le respect des délais et le travail en équipe seront des aspects essentiels de notre futur métier. De plus, il nous a permis d'appliquer nos connaissances acquises en cours et en travaux dirigés et à utiliser le langage C. Ce qui m'a aidé à mieux comprendre et à progresser. Malgré avoir eu quelques problèmes au début avec l'utilisation de la plateforme monprojet et TexMaker, j'ai pu m'investir dans le projet.

10.2.2 Thibault SAURON

J'ai beaucoup aimé travailler sur ce projet. Il m'a permis de me mettre dans une situation concrète de développement avec une équipe, un manager, des objectifs et une méthode précise. J'ai aussi pu répondre à beaucoup de questions que je me posais sur le développement d'un projet. Notamment le problème de gestion de versions que j'avais rencontré pendant le projet informatique en STPI2 ou produire quelque chose de réellement utile et que je pourrais réutiliser dans des projets futurs. J'ai aussi pu me familiariser avec de nouveaux outils comme LaTeX et ddd qui me serons utiles plus tard.

10.2.3 Matthias SESBOÜÉ

Ce projet nous a permis de mettre en application ce que nous avons appris tout au long du semestre de manière très intéressante. En plus d'appliquer nos connaissances en programmation et en

algorithmique, il nous a aussi permis de découvrir ce que c'est de travailler en groupe sur un projet informatique plus conséquent que celui sur lequel nous avions pu travailler en STPI. Enfin, nous avons découvert l'utilisation du git, outil indispensable pour travailler sur une projet informatique en équipe. En temps que chef de projet, j'ai beaucoup apprécié pouvoir "manager" une équipe dynamique constituée de personnes aux qualités et compétences variées. Ces dernières ont d'ailleurs entraîné des échanges intéressants, qui ont abouti à des choix concrets et efficaces en plus de permettre à chacun de progresser. Nous avons tous travaillé de façon plus ou moins régulière, avec chacun ses points forts et ses points faibles. Malgré quelques différences de niveau et d'investissement dans ce projet, je pense que nous nous sommes tous investi et avons mis un maximum de volonté dans sa réussite.

10.2.4 Damien TOOMEY

Ce projet m'a permis d'apprendre à utiliser le langage C. Je me suis rendu compte que le Pascal, enseigné en département STPI, est une étape nécessaire pour comprendre la base de la programmation car le C est un langage plus compliqué à manipuler. Ce projet m'a plu car contrairement aux conditions d'un examen théorique ou pratique, j'ai pu y consacrer tout le temps que je voulais. L'utilisation de la plateforme monprojet a facilité le développement du code. En revanche, je trouve que l'utilisation des demandes peut freiner la prise d'initiatives.

11 Répartition du travail

11.1 Répartition générale

Tâches	BOUTAYEB Dounia	SAURON Thibault	SESBOÜÉ Matthias	TOOMEY Damien
TAD Mot	X	-	X	-
TAD Dictionnaire	X	X	X	X
partie privée Dictionnaire	-	-	-	X
TAD CorrecteurOrthographique	-	-	X	-
partie privée CO	-	-	X	-
TAD arbre n-aire	-	X	-	-
TAD ListeChaineeMot	X	-	-	-
TAD Ensemble	-	-	X	-
partie privée Ensemble	-	-	X	-
programme principal	-	-	-	X
procédures d'affichage	-	-	-	X
ExisteFichier	-	-	-	X
Tests unitaires TAD Mot	-	-	-	X
Tests unitaires TAD Dictionnaire	-	X	-	-
Tests unitaires TAD CO	-	X	X	X
Tests unitaires TAD Arbre n-aire	-	X	-	-
Tests unitaires TAD LCMot	-	-	-	X
Tests unitaires TAD Ensemble	-	-	-	X

11.2 Travaux communs

Tâches	BOUTAYEB Dounia	SAURON Thibault	SESBOÜÉ Matthias	TOOMEY Damien
Rapport	X	X	X	X
Analyse descendante	X	X	X	X
D.estPresent	-	-	X	-
D.insererMot	-	-	X	X