|  |  |
| --- | --- |
| LogoPoly_Genie1Classe.png | École Polytechnique de Montréal  Département de Génie Informatique et Génie Logiciel |

**LOG2810**

**STRUCTURES DISCRETES**

**Hiver 2017**

**TP2 : Automates et langages**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Remis par :**

|  |  |
| --- | --- |
| **Matricule** | **Prénom & Nom** |
| **1798345** | **Adam Martin-Côté** |
| **1786724** | **Jonathan Gervais-Ranger** |
| **1828978** | **Louis Cormier** |

**À :**

**David Johannès,**

**Mariam Tagmouti**

**Le 6 avril 2017**

Table des matières

1. [Introduction 3](#_Toc478824510)
2. [Présentation de la solution 3](#_Toc478824511)

[Diagramme de classes 3](#_Toc478824512)

[Implémentation 3](#_Toc478824513)

[Implémentation des classes 3](#_Toc478824514)

1. [Difficultés rencontrées 3](#_Toc478824515)
2. [Conclusion 3](#_Toc478824516)

# Introduction

Dans le cadre du cours *Structures discrètes*, nous avons étudier plusieurs outils mathématiques importants en informatique. Une question que plusieurs personnes se posent dans le domaine informatique est comment structurer de l’informations afin de parvenir à faire une recherche plus efficace parmi les données.

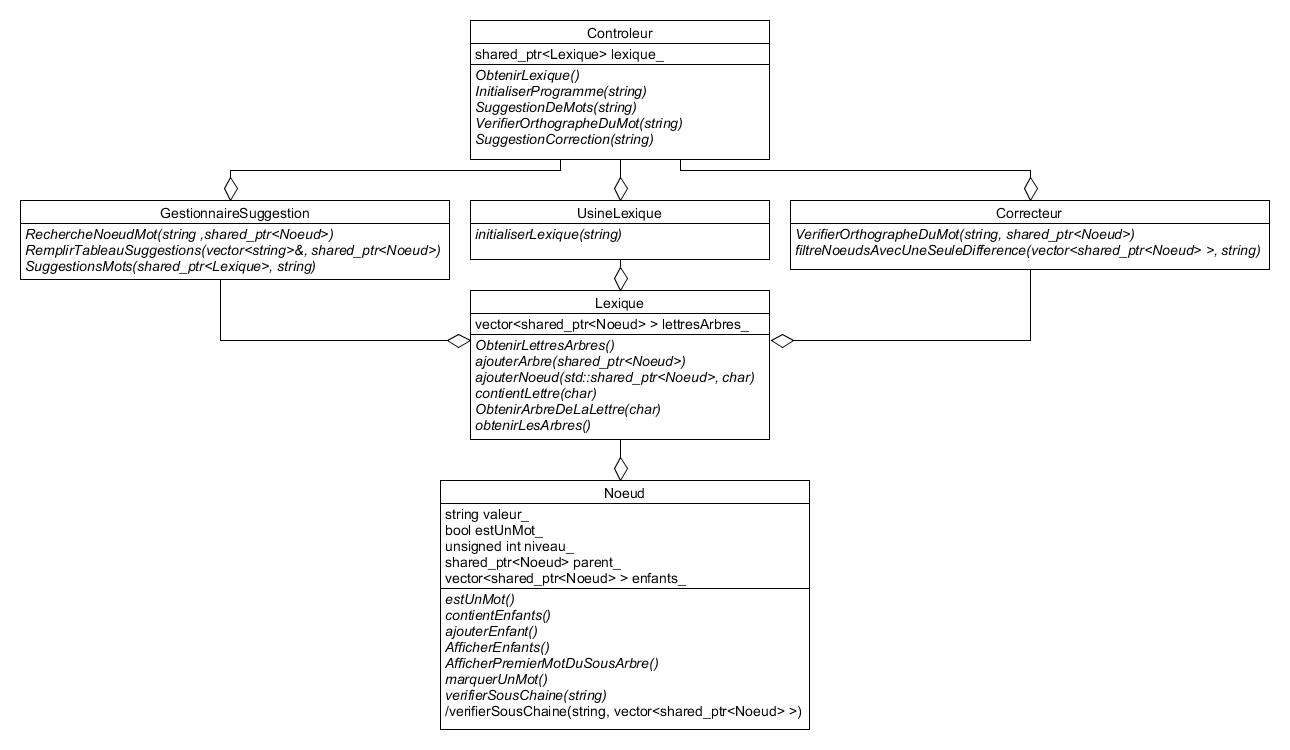
Parmi toutes les structures de données, il y en a une en particulier que nous avons étudier dernièrement, les automates. Ces derniers sont très utilisés afin de bâtir des grammaires. Ce sont de petites machines qui indique vrai si un mot fait partie d’une certaine grammaire et indique l’inverse s’il n’en fait pas partie. Puisqu’ils sont très efficaces, il nous a été demandé d’implémenter un automate qui est capable d’importer un lexique et qui permet par la suite de faire de la complétion et de la correction de mots.

Dans ce rapport, vous trouverez des informations concernant notre implémentation de ce logiciel. Plus précisément, vous trouverez dans un premier temps, vous trouverez l’explication de notre solution incluant notre diagramme de classes. Dans un deuxième temps, vous trouverez de l’information concernant l’implémentation de nos classes. Dans un troisième et dernier temps, vous trouverez de l’information concernant les difficultés que nous avons rencontrées lors de la création de notre logiciel.

# Présentation de la solution

## Diagramme de classes

Afin de parvenir d’arriver à notre solution, nous avons dû diviser le programme en quelques classes. La figure 1 présente ces dernières.



*Figure 1 : Diagramme de de classe*

Comme la figure 1 le montre bien, nous avons d’abord implémenter une classe Controleur. La fonction de cette dernière est d’appeler les méthodes les méthodes *initialiserLexique*, *suggestionsMots* et *filtreNoeudsAvecUneDifference*. Pour y parvenir, elle doit inclure les classes UsineLexique, GestionnaireSuggestion et Correcteur.

Pour arriver à importer des lexiques, nous avons implémenté une classe qui se nomme UsineLexique. Cette dernière sert à produire un lexique à partir d’un fichier.

Pour la fonctionnalité suggestion, nous avons implémenté une classe qui se nomme GestionnaireSuggestion. Cette dernière a une fonction SuggestionMots qui prend en attribut un mot et qui retourne un vecteur contenant une liste de suggestions. Cette dernière fait appel aux fonctions RechercheNoeudMot qui vérifie s’il existe un nœud dans le lexique contenant le mot entré et une autre appelé RemplirTableauSuggestions qui prend un vecteur en attribut ainsi qu’un nœud de départ et retourne un tableau rempli de suggestions.

## Implémentation

### Implémentation des classes

Pour implémenter la classe GestionnaireSuggestion, nous avons tout d’abord commencé par créer la fonction SuggestionMots qui prend en paramètre le lexique dans lequel il doit chercher ainsi que le mot entré par l’utilisateur. La première chose que cette fonction fait c’est de vérifier si le mot entré par l’utilisateur fait partie du lexique. S’il ne le trouve pas, il indiquera qu’il n’y a pas de suggestions, sinon, il continu.

Une fois que le nœud contenant le mot entré par l’utilisateur a été trouvé, nous avions besoin d’une fonction récursive capable de trouver à travers le lexique les mots existant dans les nœuds du sous-graphe commençant par le nœud trouvé. Cette fonction, prend en paramètre un vecteur par référence afin que ce soit toujours le même tableau qui est modifié et non une copie à chaque fois et le nœud de départ. Ensuite, il parcourt chacune des branches en profondeur et à chaque fois qu’il rencontre un nœud qui est un mot, il l’ajoute dans le tableau. Finalement, lorsqu’il a fini de parcourir le sous arbre ou que le tableau contient dix suggestions, il quitte la fonction et les retournes.

# Difficultés rencontrées

Une première difficulté que nous avons eu a été causé par la séparation de tâches. Il fallait que nous trouvions une façon de tester nos fonctions pendant que la personne en charge finisse sa fonction pour importer un lexique. Pour pallier à cette difficulté, nous avons créé un petit lexique temporaire.

Lors de l’implémentation de la fonction suggestion, nous avons rencontré deux difficultés. La première était de trouver une façon de parvenir à ce que le terminal se mettre à jour à chaque fois que l’utilisateur change le texte écrit. La stratégie que nous avons utilisée pour contrer cette difficulté, a été de créer une fonction où l’utilisateur doit appuyer sur la touche retour pour obtenir ces suggestions. Lorsque nous étions certains que la fonction était fonctionnelle, c’est à ce moment-là que nous nous sommes attarder à la mise à jour dans le terminal. Nous avons pris la décision de faire une interface graphique à l’aide du logiciel QT puisqu’il offre une fonctionnalité qui appelle une fonction lorsqu’une boite de texte est modifiée.

La deuxième difficulté pour la fonction suggestion a été de trouver une fonction récursive qui se promène dans le lexique. Afin de simplifier la tâche, nous avons tout d’abord implémenter une fonction qui cherche un nœud contenant le terme entré par l’utilisateur. En faisant cela, la recherche de suggestions devenait plus optimale puisqu’il aura juste à chercher des suggestions dans ce sous-graphe. De plus, en faisant cela, on pouvait aussi déjà retourner à l’utilisateur qu’il n’y a pas de suggestions puisque le nœud qu’il a entré n’existe pas. Par après, nous avons lancé une autre fonction qui se promène en profondeur dans le sous-graphe afin d’arriver à trouver tous les nœuds terminaux et qu’il arrête au moment qu’il a trouvé dix suggestions ou que le sous-graphe a fini d’être parcouru.

//AJOUTER AUTRES FONCTIONS

# Conclusion