POLYTECHNIQUE MONTRÉAL
LE GÉNIE EN PREMIÈRE CLASSE

LOG 2810
STRUCTURES DISCRETES

TP2 : AUTOMATES ET LANGAGES

Session	Hiver 2017
Pondération	10 % de la note finale
Lieu de réalisation	L-4708
Taille des équipes	3 étudiants
Date de remise du projet	5 avril 2017 (Gr. 1) et 6 avril 2017 (Gr. 2) (23h55 au plus tard)
Directives particulières	Soumission du livrable par moodle uniquement (https://moodle.polymtl.ca).
	Toute soumission du livrable en retard est pénalisée à raison de 10% par jour de retard.
	Programmation $C++$ avec Visual Studio 2013.
Les questions sont les bienvenues et peuvent être envoyées à: David Johannès (david.johannes@polymtl.ca), Mariam Tagmouti (mariam.tagmouti@polymtl.ca), John Mullins (john.mullins@polymtl.ca).	

1 Connaissances requises

- Notions d'algorithmique et de programmation C++.
- Notions d'automates et de langages.

2 Objectif

L'objectif de ce travail pratique est de vous permettre d'appliquer les notions théoriques sur les automates que nous avons vues en cours, sur des cas concrets mais hypothétiques, tirés de votre quotidien. A cet effet, il est question dans ce travail d'utiliser des automates pour encoder un lexique, faire de la complétion et de la correction de mots.

3 Mise en situation

L'ensemble des mots d'un langage forme un lexique. Dans les applications de correction automatique, le lexique peut être organisé sous la forme d'un automate. Le premier objectif est ici de construire le plus petit automate à états finis possible (minimal) qui accepte le langage d'un lexique. Le deuxième objectif est d'utiliser cet automate pour faire la complétion et la correction de mots d'une application offrant un service de correction automatique. Il s'agit donc d'implanter deux des fonctionnalités d'un correcteur automatique en utilisant l'automate reconnaissant le lexique. Ce premier automate sera donc utilisé pour deux fonctionnalités : énumérer toutes les formes possibles d'un mot donné (en donnant les premières lettres), et corriger un mot donné. Dans les cas de la correction d'un mot, nous

nous plaçons dans une situation particulière : le mot à corriger possède le même nombre de lettres que le mot qui doit le remplacer, (par exemple, le mot *bonjiur* doit être remplacé par le mot *bonjour*), et ces deux derniers mots doivent différer que d'une seule lettre.
Vous êtes sollicités pour travailler sur le développement d'une application contenant un éditeur de texte. Il vous est assigné comme travail de livrer une interface avec quelques fonctionnalités. Elle doit permettre à l'utilisateur de saisir du texte (à partir du clavier). Au fur et à mesure que l'utilisateur écrit chaque lettre (sauf le séparateur *espace* et les signes de ponctuation), votre application doit lui afficher, sur la base du lexique (automate), toutes les formes possibles du mot déjà constitué. Par exemple, sur la base du lexique dont l'extrait est présenté dans la table **??**, dès que l'utilisateur aura écrit la lettre *c*, tous les mots figurant dans l'extrait doivent lui être affichés. Dès qu'il aura écrit successivement les lettres *c*, *a*, *s*, les mots *case*, *cases*, *caser*, *casier*, *casiers* doivent lui être suggérer, et donc s'afficher à l'écran.
Aussi, votre interface doit permettre, lorsque l'utilisateur saisi du texte, de pouvoir corriger les mots mal orthographiés (en considérant le cas particulier établi dans ce TP) dès qu'il aura tapé le séparateur *espace* ou un signe de ponctuation juste après avoir écrit le mot mal orthographié. Concernant cette correction, voici comment elle peut procéder :

- Soit w, un mot du lexique, on peut construire un automate $Lev(w)$ qui reconnaisse tous les mots qui ne diffèrent de w que d'une et une seule lettre. On obtient une premi version de la correction en construisant les automates $Lev(w)$, pour tous les mots w du lexique: sur un entrée x, le programme retourne w si w est dans le lexique et si x ne diffère de w que d'une et une seule lettre. Avec un gros lexique, il est probable que votre programme prenne beaucoup de temps, même s'il est suffisant de se limiter aux mots du lexique de la même longueur que x.

- Une autre solution serait, pour une entrée x, de construire $Lev(x)$ et de retourner les mots du lexique (de la même longueur que x) qui ne diffèrent de x que d'une et une seule lettre.

- Votre automate pourrait compter le nombre d'erreurs (à savoir de lettres qui diffèrent) entre le mot mal orthographié, et les autres mots du lexique, en s'arrêtant dans le parcours (en largeur) de l'automate dès que le nombre d'erreurs entre le mot mal orthographié et l'état courant dans l'automate dépasse 1.

- Vous pouvez utiliser un automate de Mealy, qui possède sur chaque arc une étiquette d'entrée / sortie, libre à votre imagination de l'adapter à notre situation.

- Votre but, pour la fonctionnalité de correction, est de la rendre la plus performante possible. À vous de choisir la bonne méthode (peut-être une combinaison des solutions proposées, ou une autre solution qui se base bien évidemment sur l'automate minimal du lexique).

- Si plusieurs mots du lexique sont trouvés par votre automate (plusieurs mots qui diffèrent du mots mal orthographié d'une seule lettre), vous pouvez soit considérer que le mot qui doit remplacer le mot mal orthographié commence par la même lettre que celui-ci (puisqu'il est plutôt rare de se tromper sur la première lettre du mot que l'on écrit), soit proposer à l'utilisateur les mots qui pourraient remplacer le mot mal orthographié (l'utilisateur devra donc choisir parmi ces mots proposés celui qui doit remplacer le mot qu'il a mal orthographié).

- Une autre optimisation possible serait de supposer que la lettre incorrecte est une voisine de clavier de la correcte (pour le mot mal orthographié).

Remarques : comme nous utilisons les mots de la langue française, l'alphabet à considérer pour les automates est celui de la langue française (26 lettres) ; aussi, nous ne considérons pas dans notre cas les lettres accentuées. Concernant les lexiques à intégrer dans votre programme, ils vous seront fournis par l'instructeur du laboratoire, et seront par ordre croissant de de taille. Vous aurez donc à votre disposition un lexique de taille 100, un de taille 500, un de taille 1000, un de taille 2000, un de taille 5000, et un dernier de taille environ 20000 (pour les plus courageux). À la correction, vos programmes seront en compétition, et les plus performants obtiendront la meilleure note.

Mots
caisse
caisses
caissier
caissiers
caissière
caissières
cas
case
cases
caser
casier
casiers

Table 1: Extrait d'un lexique

4 Tâches à réaliser

C1. Créer un automate à partir d'un lexique. Le lexique est donné sous la forme d'un fichier .txt (vous en avez 6 à disposition).

C2. Implémenter la fonctionnalité de suggestion qui permettra de suggérer à l'utilisateur des mots à partir des premières lettres d'un mot qu'il écrit.

C3. Implémenter la fonctionnalité de correction qui permettra de corriger un mot mal orthographié par l'utilisateur (en vous plaçant dans le contexte précis où le mot mal orthographié diffère d'une seule lettre du mot qui doit le remplacer).

C4. Créer une interface graphique qui permet à l'utilisateur de saisir du texte. Au fur et à mesure que l'utilisateur écrit un mot, votre application doit lui afficher, sur la base du lexique (automate), toutes les formes possibles du mot. Par exemple, sur la base du lexique dont l'extrait est présenté ci-haut, dès que l'utilisateur écrit la lettres *c*, tous les mots figurant dans l'extrait doivent lui être affichés. Aussi, dès que l'utilisateur finit d'écrire un mot, si ce dernier est mal orthographié, il devra être remplacé par le bon mot.

C5. Faire une interface qui affiche le menu suivant :

(a) Initialiser le programme (qui prend en entrée le lexique fournit par le laboratoire et construit l'automate minimal associé).

(b) Fonctionnalité suggestion (qui permet à l'utilisateur de saisir du texte et de voir apparaître les suggestions de mots lorsqu'il écrit). Pour les lexiques les plus importants, vous vous limiterez à l'affichage des 10 premiers mots que votre programme trouvera.

(c) Fonctionnalité correction (qui permet à l'utilisateur de saisir du texte et de voir les mots mal orthographiés être corrigés).

(d) Fonctionnalité suggestion + correction (qui combine les deux dernières fonctionnalités).

(e) Retour (qui permet à l'utilisateur de revenir au menu principal à partir d'une des trois interfaces précédemment citées).

(f) Quitter (qui permet à l'utilisateur de quitter l'application à partir du menu principal).

5 Livrable

Le livrable attendu est constitué du code source et du rapport de laboratoire. Le livrable est une archive (ZIP ou RAR) dont le nom est formé des numéros de matricule des membres de l'équipe, séparés par un trait de soulignement (_). L'archive contiendra les fichiers suivants :

- les fichiers .cpp;
- les fichiers .h le cas échéant;
- le rapport au format PDF.

L'archive ne doit pas contenir de programme exécutable, de fichier de projet ou solution de Visual Studio, de répertoire Debug ou Release, etc. Les fichiers .cpp et .h suffiront pour l'évaluation du travail.

5.1 Rapport

Un rapport de laboratoire rédigé avec soin est requis à la soumission (format .pdf, maximum 8 pages). Sinon, votre travail ne sera pas corrigé (aussi bien le code source que l'exécutable). Le rapport doit obligatoirement inclure les éléments ou sections suivantes :

1. Page présentation : elle doit contenir le libellé du cours, le numéro et l'identification du TP, la date de remise, les matricules et noms des membres de l'équipe. Vous pouvez compléter la page présentation qui vous est fournie.
2. Introduction avec vos propres mots pour mettre en évidence le contexte et les objectifs du TP.
3. Présentation de vos travaux : une explication de votre solution.
4. Difficultés rencontrées lors de l'élaboration du TP et les éventuelles solutions apportées.
5. Conclusion : expliquez en quoi ce laboratoire vous a été utile, ce que vous avez appris, ce que vous voudriez approfondir dans vos projets futurs, etc.

Notez que vous ne devez pas mettre le code source dans le rapport.

5.2 Soumission du livrable

La soumission doit se faire uniquement par moodle.

6 Évaluation

Éléments évalués	Points
Qualité du rapport : respect des exigences du rapport, qualité de la présentation des solutions	1.5
Qualité du programme: compilation, structures de données, gestion adéquate des variables et de toute ressource (création, utilisation, libération), passage d'arguments, gestion des erreurs, documentation du code, etc.	1.5
Composants implémentés : respect des requis, logique de développement, etc.	
C1	3
C2	4
C3	4
C4	3
C5	3
Total de points	20

7 Documentation.txt

- http://www.cplusplus.com/doc/tutorial/
- http://public.enst-bretagne.fr/∼brunet/tutcpp/Tutoriel%20de%20C++.pdf
- Automate de Mealy : `https://fr.wikipedia.org/wiki/Machine_de_Mealy`