Compte-rendu - TP1 - CPP

Lucas Oros

Introduction du TP

L'objectif de ce TP est d'appliquer les principes de la structuration des données pour mettre en œuvre un programme de traitement de fichiers. La taille importante des fichiers considérés impose que la conception des algorithmes des programmes demandés conduise au choix des structures les plus adaptées pour éviter d'en arriver à des temps d'exécution excessifs, voire rédhibitoires.

1 CRÉATION D'UN LEXIQUE

1.1 CARACTÉRISTIQUES D'UN LEXIQUE

Nous cherchons dans un premier temps à créer un lexique de tous les mots contenus dans un texte donné. Vous créerez une **classe lexique** caractérisée par un **nom** et pour chaque mot du lexique, son nombre **d'occurrences**. Choisissez le conteneur qui vous semble le plus pertinent. Le lexique devra être constitué à partir d'un texte contenu dans un fichier d'entrée.

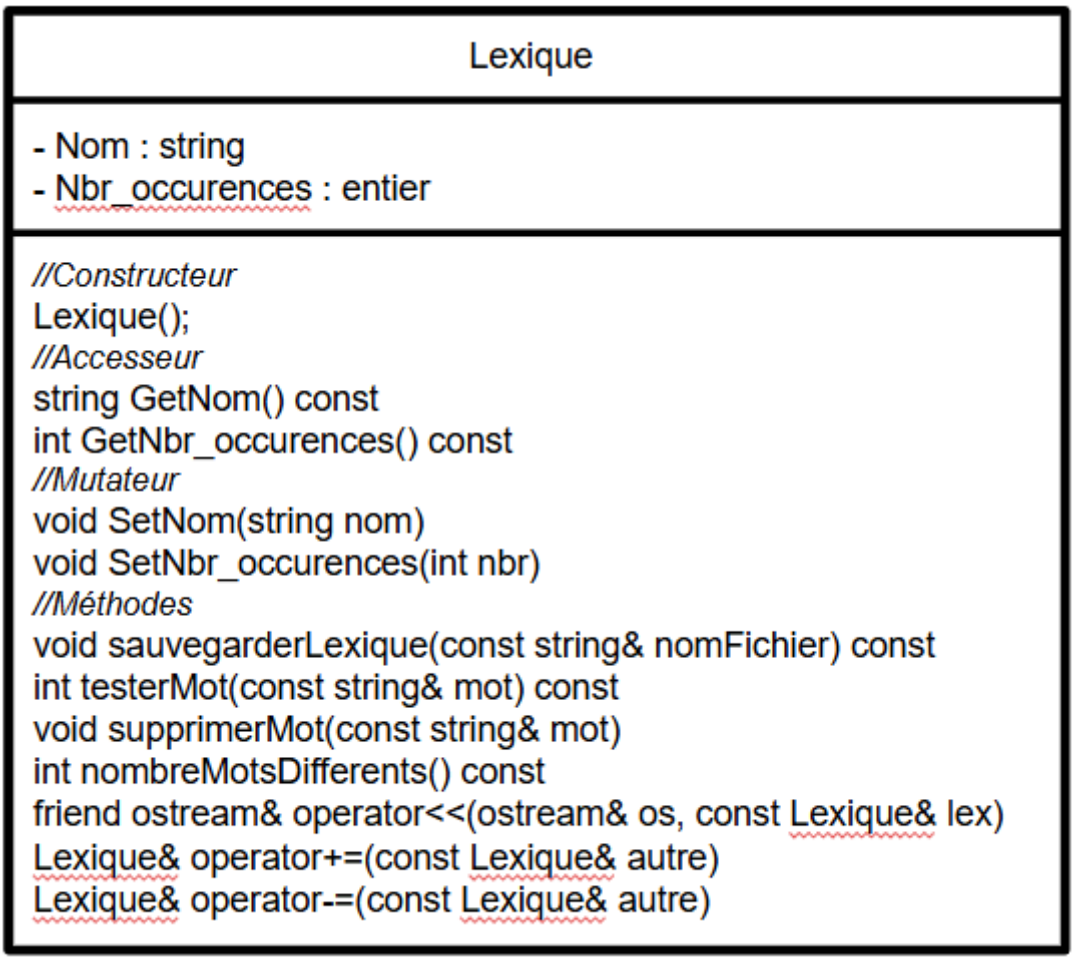
Pour cela, nous créons une classe "lexique" dans le fichier lexique.h.

1.2 OPÉRATION SUR UN LEXIQUE

Nous devons implémenter quatre(4) méthodes de la classe lexique :

- sauvegarder le contenu du lexique ainsi produit dans un fichier de sortie;
- tester la présence d'un mot dans le lexique et retourner son nombre d'occurrences;
- supprimer un mot du lexique;
- afficher le nombre de mots différents présents dans le lexique

Voici le diagramme de classe de la classe lexique :



Nous allons faire l'algorithme d'une de ces quatre(4) fonctions avec :
Fonction supprimerMot

Algorithme du programme

Fonction supprimerMot
fonction supprimerMot
paramètre : mot (chaîne de caractères)
résultat : met à jour le fichier du lexique sans le mot supprimé

Algorithme
début

Ouvrir le fichier d'entrée correspondant au lexique
si le fichier ne peut pas être ouvert
    afficher un message d'erreur et arrêter la fonction.

Créer une variable nouveauTexte (flux texte) pour stocker le contenu filtré.
Nettoyer le mot à supprimer :
    retirer la ponctuation
    mettre en minuscules

Initialiser :
    motSupprime ← faux
    ligne (chaîne vide)

tant que le fichier contient une ligne à lire :
    Séparer la ligne en mots (motActuel).
    Pour chaque mot de la ligne :
        Nettoyer le mot (motTest).
        si motTest = motNettoye
            marquer motSupprime ← vrai
            ne pas recopier le mot.
        sinon, ajouter le mot à ligneFiltree avec un espace si nécessaire.
    Ajouter la ligneFiltree au flux nouveauTexte et passer à la ligne suivante.

Fermer le fichier d'entrée.

si motSupprime est encore faux
    afficher "Le mot n'est pas présent dans le lexique" et arrêter la fonction.

Ouvrir le fichier de sortie en mode écrasement.

si le fichier ne peut pas être ouvert
    afficher un message d'erreur et arrêter la fonction.

Écrire dans le fichier le contenu de nouveauTexte.
Fermer le fichier de sortie.
Afficher un message confirmant la suppression
Fin

Nous allons également faire l'algorithme avec la fonction de surcharge: += pour fusionner deux lexiques (opérateur interne à la classe Lexique);

Algorithme du programme

Fonction operator+=
But : fusionner deux lexiques
Paramètres : lexique courant (le lexique récepteur), autre (le lexique à fusionner)
Résultat : le lexique courant contient la fusion des deux lexiques, avec cumul des occurrences

Algorithme
début

Déterminer le nom des fichiers :
    fichier1 ← "tp1-Lexique-output/Lexique-" + Nom + "-output.txt"
    fichier2 ← "tp1-Lexique-output/Lexique-" + autre.GetNom() + "-output.txt"

Ouvrir les deux fichiers (in1 et in2) en lecture.
    si l'un des deux fichiers ne peut pas être ouvert
        afficher un message d'erreur et arrêter la fonction.

Créer une map occurrences vide pour stocker les mots et leur nombre total d'apparitions.

Définir une fonction locale lireFichier(fichier) qui :
    lit chaque ligne du fichier ;
    découpe la ligne en mots ;
    nettoie chaque mot (remove_punctuation, to_lower);
    si le mot est non vide → incrémente occurrences[mot].

Appeler :
    lireFichier(in1) pour le premier lexique ;
    lireFichier(in2) pour le second lexique.

Fermer les deux fichiers d'entrée.
Ouvrir fichier1 en écriture (mode troncature pour écraser l'ancien contenu).
    si le fichier ne peut pas être ouvert
        afficher une erreur et arrêter la fonction.

Pour chaque couple (mot, count) dans occurrences :
écrire dans fichier1 : mot (count)

Fermer le fichier de sortie.
Afficher un message de confirmation
Retourner le lexique courant.

fin

Nous allons terminé par faire l'algorithme avec la fonction de surcharge: -= pour faire la différence de deux lexiques, ie. ne garder que les mots dans le premier lexique qui ne sont pas dans le deuxième (opérateur interne à la classe Lexique).

Algorithme du programme

Fonction operator-=
But : effectuer la différence entre deux lexiques
Paramètres : lexique courant (le lexique récepteur), autre (le lexique à comparer)
Résultat : le fichier du premier lexique est réécrit sans les mots présents dans le second

Algorithme
début
Déterminer les noms des fichiers associés aux deux lexiques :
    fichier1 ← "tp1-Lexique-output/Lexique-" + Nom + "-output.txt"
    fichier2 ← "tp1-Lexique-output/Lexique-" + autre.GetNom() + "-output.txt"

Ouvrir les deux fichiers (in1 et in2) en lecture.
    si un des fichiers ne peut pas être ouvert
        afficher une erreur et arrêter la fonction.

Créer un ensemble motsASupprimer (de type set<string>) pour stocker les mots présents dans le second lexique.

Lire chaque ligne du fichier in2 :
    Découper la ligne en mots.
    Nettoyer chaque mot (remove_punctuation, to_lower).
    si le mot n'est pas vide, l'ajouter à motsASupprimer.

Créer un ensemble motsResultats vide pour les mots à conserver.
Lire chaque ligne du premier lexique (in1) :
    Découper la ligne en mots.
    Nettoyer chaque mot (remove_punctuation, to_lower).
    si le mot est non vide et n'appartient pas à motsASupprimer
        l'ajouter à motsResultats.

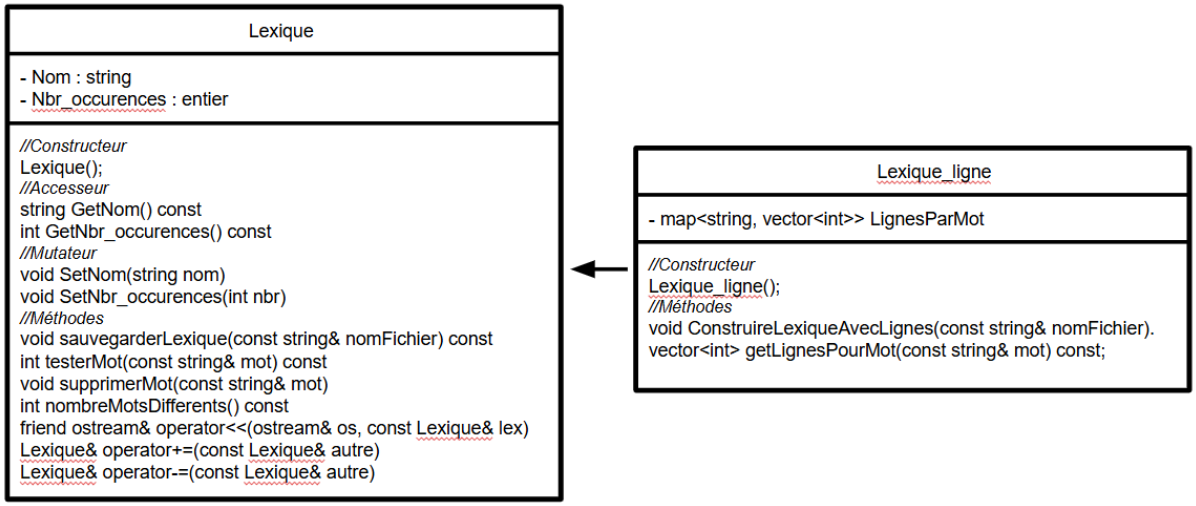
Fermer les deux fichiers d'entrée.
Ouvrir fichier1 en écriture (mode troncature pour effacer le contenu précédent).
    si l'ouverture échoue
        afficher une erreur et arrêter la fonction.

Pour chaque mot m dans motsResultats :
    écrire m suivi d'un retour à la ligne.

Fermer le fichier de sortie.
Afficher un message de confirmation
Retourner le lexique courant.
fin

2 CRÉATION D'UN LEXIQUE AVEC NUMÉRO DE LIGNE

Nous devons créer ensuite un lexique qui héritera de votre première classe **lexique** mais qui permet de garder en plus pour chaque mot du lexique la ligne du fichier où il a été rencontré.
Voici le diagramme de classe de la classe Lexique_Ligne hérité de la classe Lexique:



3 Jeux d'essais

Nous avons dans le fichier test.h et test.cpp un ensemble de jeux d'essais en fonction des méthodes implémentées dans les classes.
void test_partie1_methode1(void);
void test_partie1_methode2(void);
void test_partie1_methode3(void);
void test_partie1_methode4(void);
void test_partie1_surcharge(void);
void test_partie2(void);
Il suffit donc d'appeler une des fonctions pour tester le code.

Tester la présence d'un mot dans le lexique et retourner son nombre d'occurrences

Pour cette méthode, nous avons choisi de travailler à partir d'un texte court d'exemple nommé LexiqueDeLucas_A.txt, placé dans le dossier d'entrée tp1-Lexique-input.
Lors de l'exécution du test, le programme crée automatiquement un fichier de sortie (Lexique-MonLexique_partie2-output.txt) contenant le lexique du texte, c'est-à-dire la liste de tous les mots présents après nettoyage et mise en forme.

La fonction testerMot() permet alors de vérifier si un mot donné figure dans ce lexique et d'en compter le nombre d'occurrences.
Par exemple, pour le mot « bonjour », que l'on sait apparaître quatre fois dans le texte d'origine, l'appel de la fonction confirme bien cette information : le programme affiche que le mot bonjour est présent avec 4 occurrences dans le lexique.

```
Lancement de l'exécutable
.\TP1exe.exe
Lexique : <MonLexique_partie2> a bien ete cree.
Le lexique a ete sauvegarde dans : tp1-Lexique-output/Lexique-MonLexique_partie2-output.txt
Entrez un mot a tester dans le lexique : bonjour
Le mot <bonjour> est present dans le lexique avec 4 occurrences.
```

Afficher le nombre de mots différents présents dans le lexique

Cette méthode a pour objectif de déterminer le nombre total de mots distincts contenus dans le texte.
Pour cela, le programme lit le fichier d'entrée et construit le lexique correspondant, en supprimant les doublons et en ne conservant qu'une seule occurrence de chaque mot unique.

Afin de valider le fonctionnement, nous avons comparé le résultat obtenu avec la valeur de référence trouvée sur un site d'analyse de texte, qui indique le nombre exact de mots différents contenus dans le document, ici 41.
Lors de l'exécution, la fonction renvoie bien cette même valeur, confirmant ainsi la fiabilité de la méthode et la bonne prise en compte du nettoyage et de la normalisation des mots dans le lexique, avec en plus le lexique des mots dans le terminal avec le nombre d'occurence de chaque mot.

Test des surcharges d'opérateurs

Cette partie du TP vise à tester le bon fonctionnement des opérateurs surchargés de la classe Lexique, en particulier les opérateurs affichage, += (fusion) et -= (différence).

Pour cela, deux lexiques distincts sont d'abord créés à partir des fichiers d'entrée LexiqueDeLucas_A.txt et LexiqueDeLucas_B.txt.
Chacun d'eux est sauvegardé dans le dossier tp1-Lexique-output sous les noms :

Lexique-MonLexique_surcharge_A-output.txt

Lexique-MonLexique_surcharge_B-output.txt

L'opérateur += est ensuite appelé afin de fusionner les deux lexiques :
le premier lexique reçoit l'ensemble des mots des deux fichiers, et les occurrences de chaque mot sont additionnées.
Le résultat affiché à l'écran montre bien la fusion complète, confirmant que les deux lexiques ont été combinés correctement.

De la même manière, l'opérateur -= (non exécuté ici mais également implémenté) permettrait de soustraire les mots communs entre les deux lexiques, ne laissant dans le premier que les mots absents du second.

```
pour : 1
proches : 1
programmation : 1
programme : 1
programmeur : 1
quelques : 1
qu’une : 1
retrouve : 1
rivière : 1
répétés : 1
second : 1
seule : 1
similaires : 1
test : 1
test1 : 1
test2 : 1
test3 : 1
tester : 1
testeur : 1
testing : 1
tests : 1
texte : 1
tous : 1
un : 1
une : 1
uniques : 1
variantes : 1
voici : 1
y : 1
à : 1
Nombre total de mots differents : 80
```

Lexique hérité : ajout des lignes d'apparition des mots

Dans cette dernière partie, nous avons créé une nouvelle classe appelée Lexique_Ligne, qui hérite de la classe de base Lexique.
Cette classe enrichit les fonctionnalités existantes en ajoutant une information supplémentaire : pour chaque mot du lexique, le programme enregistre les numéros de lignes dans lesquelles ce mot apparaît dans le texte d'origine.

Concrètement, lors de la lecture du fichier texte, la méthode ConstruireLexiqueAvecLignes() parcourt le contenu ligne par ligne à l'aide de la fonction getline() (de la bibliothèque <string>).
À chaque ligne, le texte est découpé en mots, nettoyé (ponctuation supprimée, mise en minuscules), puis stocké dans une structure de type map<string, vector<int>>.
Ainsi, à chaque mot correspond un vecteur d'entiers contenant les numéros de lignes où il a été rencontré.

Lors du test (test_partie2()), l'utilisateur saisit un mot dans la console :
le programme affiche alors la liste des lignes où ce mot est présent.

```
PS C:\Users\Lucas\Desktop\Centrale TP\POO\tp1-tp1-djamael-et-lucas\TP1-Lucas> make run
g++ -o TP1exe main.o lexique.o lexique_ligne.o utilitaire.o test.o
Activation de l'encodage UTF-8 pour le terminal...
Lancement de l'exécutable
.\TP1exe.exe
Le lexique a ete sauvegarde dans : tp1-Lexique-output/Lexique-MonLexique_herite-output.txt
Lexique herite : <MonLexique_herite> a bien ete cree.
Entrez un mot a tester dans le lexique herite : bonjour
Le mot <bonjour> est present dans le lexique herite aux lignes : 1 21 22 25
PS C:\Users\Lucas\Desktop\Centrale TP\POO\tp1-tp1-djamael-et-lucas\TP1-Lucas>
```