
TP1 CPP

But :

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.

I. Réalisations des programmes

1. Création d'un lexique:

On définit une classe Lexique chargée de stocker l'ensemble des mots d'un texte ainsi que le nombre d'occurrences de chacun. La structure de données la plus adaptée pour représenter ce lexique est une *std::map*, qui implémente un tableau associatif : chaque élément est constitué d'une clé et d'une valeur associée.

Dans notre cas, la clé correspond au mot, et la valeur correspond au nombre de fois où ce mot apparaît dans le texte.

Le diagramme de classe de la classe Lexique est représenté ci-dessous :

Lexique
- string nom - map<string, int> mots;
// Constructeurs + Lexique() + Lexique(string_view path)
// Getters + string get_nom() const + const map<string, int>& get_mots() const
// Méthodes pour les opérations sur les lexiques + void print() + bool save() + int check_word_presence(string word) + void delete_word(string word) + void print_all_words()
// Méthode pour la surcharge des opérateurs += et -= + void operator+=(Lexique const& other) + void operator-=(Lexique const& other)

La classe Lexique comporte différentes méthodes notamment des méthodes qui permettent de surcharger les opérateurs += et -= afin de respectivement permettre de fusionner deux lexiques et de faire la différence de deux lexiques.

Les algorithmes pour ces deux méthodes sont les suivantes :

Fonction operator+=

Problème : Ajouter au lexique courant tous les mots d'un autre lexique, en incrémentant les occurrences si les mots existent déjà.

Spécification

 Fonction aucun \leftarrow operator+= (Lexique other)

 Paramètre

 Lexique other

 Résultat : Aucun

Algorithme

Variables :

 it : itérateur constant parcourant les mots du lexique other

Début :

 Pour chaque paire (mot, occurrences) dans other.get_mots() faire

 Si le mot existe déjà dans this->mots alors

 Incréments this->mots[mot] de occurrences

 Sinon

 Ajouter le mot dans this->mots avec occurrences comme valeur

 FinSi

 FinPour

Fin

Fonction operator-=

Problème : Retirer du lexique courant les mots contenus dans un autre lexique, en diminuant le nombre d'occurrences. Si un mot atteint zéro (ou une valeur négative), il est supprimé du lexique.

Spécification

 Fonction aucun \leftarrow operator-= (Lexique other)

 Paramètre

 Lexique other

 Résultat : Aucun

Algorithme

Variables :

 it : itérateur constant parcourant les mots du lexique other

Début :

 Pour chaque paire (mot, occurrences) dans other.get_mots() faire

 // Soustraire les occurrences

 this->mots[mot] \leftarrow this->mots[mot] - occurrences

 // Vérifier si le mot doit être supprimé

 Si this->mots[mot] \leq 0 alors

 Supprimer le mot de this->mots

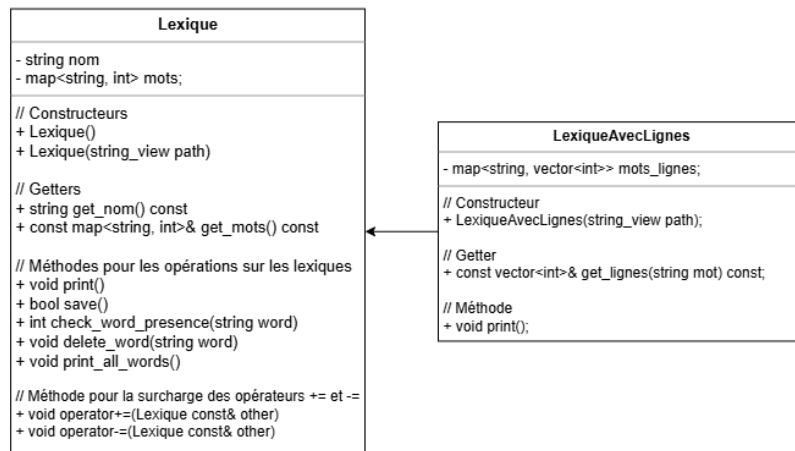
 FinSi

 FinPour

Fin

2. Création d'un lexique avec numéro de lignes :

Nous créons désormais une classe `LexiqueAvecLignes` qui hérite de la classe `Lexique` et qui va permettre de référencer les lignes où chaque mot apparaît.



Cette sous-classe contient désormais une map d'élément ayant chacun pour clef un mot et pour valeur un tableau dynamique d'entier dont le premier élément correspond à l'occurrence du mot et les autres éléments correspondent aux lignes où le mot a été rencontré.

II. Test : Jeux d'essais :

I. Lexique classique :

Nous allons désormais illustrer le bon fonctionnement de notre programme :

Pour tester mon programme je me suis basé sur deux fichiers textes plus courts que ceux proposés :

Lexique1 :

```
Fichier  Modifier  Affichage  H1  ⌵  ≡

The Project Gutenberg EBook of Les Misérables, by Victor Hugo.
You must know that we have parents. Parents--you do not know much about
such things. They are called fathers and mothers by the civil code,
which is puerile and honest. Now, these parents groan, these old folks
implore us, these good men and these good women call us prodigal sons
```

Lexique2 :

```
Fichier  Modifier  Affichage

The Project Gutenberg EBook of Les Misérables, by Victor Hugo.
```

Nous créons alors deux instances de la classe Lexique, l'une se basant sur le fichier Lexique1 et l'autre sur Lexique2. Nous testons alors les différentes opérations en se basant sur ces deux fichiers.

a) Affichage du lexique

Nous affichons les deux lexiques :

Lexique1 :

```
Lexique: tp1-Lexique-fichiers/Lexique1
Mots présents :
- about : 1
- and : 3
- are : 1
- by : 2
- call : 1
- called : 1
- civil : 1
- code : 1
- do : 1
- ebook : 1
- fathers : 1
- folks : 1
- good : 2
- groan : 1
- gutenberg : 1
- have : 1
- honest : 1
- hugo : 1
- implore : 1
- is : 1
- know : 2
- les : 1
- men : 1
- misérables : 1
- mothers : 1
- much : 1
- must : 1
- not : 1
- now : 1
```

```
- old : 1
- parents : 2
- parentsyou : 1
- prodigal : 1
- project : 1
- puerile : 1
- sons : 1
- such : 1
- that : 1
- the : 2
- these : 4
- they : 1
- things : 1
- us : 2
- victor : 1
- we : 1
- which : 1
- women : 1
- you : 1
```

Lexique2 :

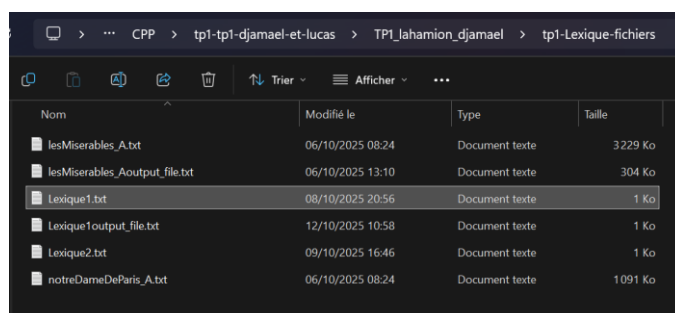
```
Lexique: tp1-Lexique-fichiers/Lexique2
Mots présents :
- by : 1
- ebook : 1
- gutenber : 1
- hugo : 1
- les : 1
- misérables : 1
- of : 1
- project : 1
- the : 1
- victor : 1
```

Nous pouvons observer que chaque lexique est bien affiché avec l'ensemble des mots qu'il contient et leur occurrences associées.

b) sauvegarde du contenu du lexique dans un fichier de sortie :

Nous testons la sauvegarde sur le premier lexique :

Nous constatons alors la création d'un fichier Lexique1outputfile.txt qui contient la liste des mots contenu dans le fichier Lexique1 avec leur occurrence.



Nom	Modifié le	Type	Taille
lesMiserables_A.txt	06/10/2025 08:24	Document texte	3229 Ko
lesMiserables_Aoutput_file.txt	06/10/2025 13:10	Document texte	304 Ko
Lexique1.txt	08/10/2025 20:56	Document texte	1 Ko
Lexique1output_file.txt	12/10/2025 10:58	Document texte	1 Ko
Lexique2.txt	09/10/2025 16:46	Document texte	1 Ko
notreDameDeParis_A.txt	06/10/2025 08:24	Document texte	1091 Ko

```

about 1
and 3
are 1
by 2
call 1
called 1
civil 1
code 1
do 1
ebook 1
fathers 1
folks 1
good 2
groan 1
gutenberg 1
have 1
honest 1
hugo 1
implore 1
is 1
know 2
les 1
men 1
misérables 1
mothers 1
much 1
must 1
not 1
now 1
of 1
old 1
parents 2
parentsyoud 1
prodigal 1
project 1
puerile 1
sons 1

```

...

On constate que le contenu du fichier de sortie correspond à l’affichage que l’on avait précédemment pour le premier lexique.

c) Vérification de la présence d’un mot

Nous testons la présence du mot « these » dans le premier lexique. Si ce mot est présent, un message indique combien de fois il est présent.

Nous obtenons alors que le message suivant.

```

Test de la présence du mot these dans le premier Lexique
Le mot est present 4 fois

```

Ceci correspond bien à ce qui est indiqué dans le lexique 1 et dans le fichier texte dont il provient.

d) Suppression d’un mot du lexique

Nous supprimons le mot « these » du lexique1 et observons de nouveau le lexique pour vérifier si le mot a bien été supprimé.

Nous observons directement sur l’affichage du lexique que le mot n’est plus présent :

```
- the : 2
- they : 1
- things : 1
```

Nous utilisons ensuite la fonction permettant de vérifier si un mot est présent dans le lexique pour nous assurer que le mot « these » a bien été supprimé :

```
Test de la présence du mot these dans le premier Lexique après suppression
Le mot n'est pas présent
```

e) Affichage du nombre de mots différents dans le lexique

Nous affichons le nombre de mots différents dans le deuxième lexique :

```
Lexique: tp1-Lexique-fichiers/Lexique2
Mots présents :
- by : 1
- ebook : 1
- gutenbergr : 1
- hugo : 1
- les : 1
- misérables : 1
- of : 1
- project : 1
- the : 1
- victor : 1

Il y a dans ce lexique 10 mots différents.
```

Le nombre affiché est bien cohérent.

f) Fusion de deux lexiques

Nous fusionnons le contenu du lexique2 avec celui du lexique1. En réaffichant le lexique 2, nous observons que le contenu du lexique 1 a été ajouté au lexique2 :


```
Fusion des lexiques
Lexique: tp1-Lexique-fichiers/Lexique2
Mots présents :
- about : 1
- and : 3
- are : 1
- by : 3
- call : 1
- called : 1
- civil : 1
- code : 1
- do : 1
- ebook : 2
- fathers : 1
- folks : 1
- good : 2
- groan : 1
- gutenberg : 2
- have : 1
- honest : 1
- hugo : 2
- implore : 1
- is : 1
```

...

Les mots déjà présents dans le lexique 2 et que l'on retrouve dans le lexique 1 ont désormais leur nombre d'occurrences qui a changé et qui correspond à la somme des occurrences du lexique 2 et ceux du lexique1.

Par exemple le mot « by », présent 2 fois dans le lexique 1 et 1 fois dans le lexique 2, est désormais présent 3 fois dans le lexique 2.

Les mots précédemment absents du lexique 2 et que l'on retrouve dans le lexique 1 sont désormais présents dans le lexique 2 avec le nombre d'occurrences qu'ils avaient dans le lexique 1.

a) Différence de deux lexiques

Nous testons la différences de lexiques en faisant la différence entre le lexique 2, nouvellement fusionné avec le lexique 1 , et le lexique 1, qui n'a subi aucune modification :

```
Difference des lexiques
Lexique: tp1-Lexique-fichiers/Lexique2
Mots présents :
- by : 1
- ebook : 1
- gutenber : 1
- hugo : 1
- les : 1
- misérables : 1
- of : 1
- project : 1
- the : 1
- victor : 1
```

Nous constatons que l'on retrouve bien le contenu du lexique 2 qui était présent au départ. On ne garde finalement que les mots présents dans le deuxième lexique mais qui ne sont pas dans le premier.

II. Lexique avec lignes :

Nous allons désormais la création des lexiques avec l'ajout des numéros de ligne.

Pour ce faire, nous créons une instance de la sous-classe `LexiqueAvecLignes` qui se basera sur le fichier texte `Lexique1`.

Nous affichons ensuite le contenu du lexique :

```

about (1 occurrences, lignes: 2)
and (3 occurrences, lignes: 3, 4, 5)
are (1 occurrences, lignes: 3)
by (2 occurrences, lignes: 1, 3)
call (1 occurrences, lignes: 5)
called (1 occurrences, lignes: 3)
civil (1 occurrences, lignes: 3)
code (1 occurrences, lignes: 3)
do (1 occurrences, lignes: 2)
ebook (1 occurrences, lignes: 1)
fathers (1 occurrences, lignes: 3)
folks (1 occurrences, lignes: 4)
good (2 occurrences, lignes: 5, 5)
groan (1 occurrences, lignes: 4)
guttenberg (1 occurrences, lignes: 1)
have (1 occurrences, lignes: 2)
honest (1 occurrences, lignes: 4)
hugo (1 occurrences, lignes: 1)
implore (1 occurrences, lignes: 5)
is (1 occurrences, lignes: 4)
know (2 occurrences, lignes: 2, 2)
les (1 occurrences, lignes: 1)
men (1 occurrences, lignes: 5)
misérables (1 occurrences, lignes: 1)
mothers (1 occurrences, lignes: 3)
much (1 occurrences, lignes: 2)
must (1 occurrences, lignes: 2)
not (1 occurrences, lignes: 2)
now (1 occurrences, lignes: 4)
of (1 occurrences, lignes: 1)

```

Nous observons que chaque mot est bien affiché avec son nombre d'occurrence suivi de la liste des lignes auxquelles on retrouve le mot. Le mot « and » par exemple est présent 3 fois dans le texte, une fois à la ligne 3, une autre fois à la ligne 4 et une dernière fois à la ligne 5. Nous observons bien cela dans le texte :

```

The Project Gutenberg EBook of Les Misérables, by Victor Hugo.
You must know that we have parents. Parents--you do not know much about
such things. They are called fathers and mothers by the civil code,
which is puerile and honest. Now, these parents groan, these old folks
implore us, these good men and these good women call us prodigal sons

```

III. Conclusion :

Ce TP a permis de mettre en pratique les concepts de structuration des données en C++ pour le traitement de fichiers textes. La réalisation des classes Lexique et LexiqueAvecLignes a montré l'importance du choix des structures de données adaptées, ici `std::map` pour associer efficacement mots et occurrences, et l'utilisation de tableaux dynamiques pour stocker les numéros de lignes.

Les différents tests réalisés ont confirmé le bon fonctionnement des opérations fondamentales : affichage, vérification de la présence d'un mot, suppression, fusion et différence de lexiques.

Ainsi, ce TP a consolidé les notions de manipulation de conteneur, d'héritage et de surcharge d'opérateurs en C++, tout en sensibilisant à l'importance de choisir des structures de données efficaces pour un traitement optimal des informations.