Présentation de la spécification et de la conception

1. **Spécification complète de l’application**

* La ligne de commande pour exécuter l’application doit être exactement de la forme d’une de ces quatre commandes :
* *TpReferencesCroisees.exe -e -k fichier\_mots\_cles [nomfichier]\**
* *TpReferencesCroisees.exe -e [nomfichier]\**
* *TpReferencesCroisees.exe -k fichier\_mots\_cles [nomfichier]\**
* *TpReferencesCroisees.exe [nomfichier]\**
* Dans le cas où un même fichier serait présent plusieurs fois dans les arguments, l'application ne réalisera le traitement qu'une seule fois.
* La présence d’un identificateur dans un fichier à analyser présent plusieurs fois sur une même ligne donne le même résultat que si l’identificateur est présent sur plusieurs fois sur une ligne différente.

Ex( pour un identificateur présent deux fois sur la 3ème ligne) :

*Cout file.cpp 3 3*

* Si le fichier fichier\_mots\_clés est vide, on considère qu’aucun mot-clé ne doit être pris en compte.
* On considère que le premier numéro de ligne d'un fichier porte l'indice 1.
* Les commentaires seront toujours pris en compte selon la syntaxe du C++, c’est-à-dire les caractères situés après « // » au sein d’une même ligne ainsi que tous les caractères situés entre « /\* » et « \*/ ».
* On exclue d’analyser les chaines de caractères ainsi que les caractères situés entre « ‘ »
* L’application considère que sur chaque ligne du fichier fichier\_mots\_cles ne comporte qu’un seul mot-clé. S’il y a présence des symboles suivants dans le fichier, le programme envoie un message d’erreur et s’arrête : *l’espace, la tabulation, la virgule, le point-virgule*. Ces symboles ont comme signification de séparation.
* Lorsque l’on écrit des arguments syntaxiquement invalides, le programme envoie un message d’erreur spécifique et s’arrête
* Lorsque l’on écrit un fichier fichier\_mots\_clés qui n’existe pas ou qui ne peut pas permettre la lecture, le programme envoie un message d’erreur et s’arrête
* Si on ne met aucun nom de fichier à analyser, le programme envoie un message d’erreur et s’arrête
* Si un ou plusieurs noms de fichiers à analyser sont invalides, le programme envoie un message d’erreur et s’arrête sans donner de résultat sur les fichiers qui auraient pu être analysés

1. **Test fonctionnels**

Test 1.1 : On positionne l’option « -e » à la fin de la commande

**Résultat attendu** : Affiche ce message : *Position de l'argument -e invalide*

Test 1.2 : On positionne l’option « -e » juste avant le premier fichier à analyser

**Résultat attendu** : Affiche ce message : *Position de l'argument -e invalide*

Test 1.3 : On positionne l’option « -e » juste après l’option « -k »

**Résultat attendu** : Affiche ce message : *FichierMotCles attendu apres l'option –k*

Test 1.4 : On positionne l’option –k après fichier\_mot\_cles

**Résultat attendu** : Affiche ce message : *Position de l'argument -k invalide*

Test 1.5 : On exécute la commande sans mettre d’arguments

**Résultat attendu** : Affiche ce message : *Nombre d'arguments insuffisant*

Test 1.6 : On exécute la commande sans indiquer de fichier à analyser

**Résultat attendu** : Nombre d'arguments insuffisant

Test 1.7 : On exécute la commande en mettant une option inconnue

**Résultat attendu** : Affiche ce message : *Option inconnue*

Test 2 : On exécute la commande suivante :

*TpReferencesCroisees.exe fileTest2.txt fileTest2.txt*

Où fileTest2.txt ne comporte que cette ligne : *int main();*

**Résultat attendu** : Affiche ce message : *int fileTest2.txt 1*

Test 3 : On exécute la commande suivante :

*TpReferencesCroisees.exe fileTest3.txt*

Où fileTest3.txt ne comporte que cette ligne : *int main();int entier ;*

**Résultat attendu** : Affiche ce message : *int fileTest3.txt 1 1*

Test 4 : On exécute la commande en indiquant comme fichier fichier\_mots\_cles un fichier vide

**Résultat attendu** : (rien ne sort de la sortie standard)

Test 5 : On exécute la commande en indiquant comme fichier à analyser un fichier comportant des « // », des « /\* » avec des « \*/ », des « ‘’ » et « ‘ »

**Résultat attendu** : (les caractères situés derrière « // » au sein d’une ligne et tous les caractères situés entre « /\* » et « \*/ » ne sont pas affichés comme indicateur. Il en va de même pour les chaines de caractères ainsi que les caractères situés entre « ‘ ».

Test 6 : On exécute la commande en indiquant comme fichier fichier\_mots\_cles un fichier comportant des tabulations, des espaces, des virgules, des Erreur

**Résultat attendu** : Affiche ce message : *Erreur*

Test 7 : On exécute la commande en indiquant un fichier fichier\_mots\_clés qui n’existe pas

**Résultat attendu** : Affiche ce message : *Erreur*

Test 8 : On exécute la commande en indiquant plusieurs fichiers à analyser, mais avec un fichier invalide

**Résultat attendu** : Affiche ce message : *Erreur*

1. **Architecture générale de l’application**
2. **Algorithmes principaux**
3. **Analyse critique**

Nous avons choisi dans de représenter l’association entre les références et les fichiers par un arbre. Cela permet d’avoir une insertion de référence beaucoup plus efficace qu’une liste chainée ou un tableau. Chaque élément de l’arbre corresponde à des clés uniques correspondant aux identificateurs qui pointent sur un objet AssocRefFichier.

Nous avons procédé de même pour l’association entre les fichiers et les lignes.