Références croisées
Spécification et Conception

1 Spécification complète

1.1 Définitions

- **ligne** : Une suite de caractères terminées par un retour chariot
- **Identificateur** : Mot sensible à la casse composé uniquement de caractères alphanumériques et du caractère '_'. Les commentaires ou chaines litérales ne peuvent contenir d'identifcateurs
- **Délimiteur** : Un caractère représentant une séparation entre deux mots (Ex : une virgule, un espace, un point, ...)
- **Référence croisée** : Fait de rechercher un identificateur dans un ou plusieurs fichiers sources pour déterminer sa localisation

1.2 Description du programme

Le but du programme est de permettre de retrouver rapidement l'emplacement d'identifcateurs dans une collection de fichiers. On cherche à connaitre dans quel(s) fichier(s) et à quelle(s) lignes les identifcateurs apparaissent.

Dans le cas où un identifacteur apparaitrait plusieurs fois sur une même ligne, nous avons pris la décision d'afficher la ligne conserné autant de fois qu'il y a d'occurences.

Ex : Pour le code ci-dessous présent dans le fichier "test.cpp" et avec comme identifacteur la lettre i

```
for( int i = 0; i < 42; i++ );
```

Le programme produira la sortie suivante :
i→test.cpp•1•1•1↓

Par defaut les identificateurs sont les mots clefs utilisés par le langage C++. Il est cependant possible de spécifier un fichier en argument du programme pour définir précisement quelles seront les identifcateurs recherchés par la référence croisée. Le fichier d'identifacteurs ne devra contenir qu'un seul identifacteur valide par ligne. Le programme présuposera que le fichier fournit en argument respecte ce formalisme.

Le programme disposera également de la fonctionnalité permettant d'exclure une liste d'identifacteurs.

1.3 Spécifications des options

$$tp\_stl\ [-e]\ [-kfichier\_mot\_clef]\ [nomfichier]+$$

-e : Permet d'inverser le comportement par défaut du programme. Exclut de la référence croisée tous les mots clefs
-k fichier_mot_clef : Permet de spécifier au programme une liste d'identifacateurs à rechercher par la référence croisée
nomfichier : Chemin vers un ou plusieurs fichiers où rechercher les identifacateurs

2 Tests fonctionnels

2.1 Méthodologie

Nous sommes parti du principe que le programme doit produire la même sortie écran (et donc par extension les mêmes données stockées en interne) si il est lancé deux fois sur la même collection de fichiers et avec les mêmes arguments.

En partant de ce principe nous avons réaliser les tests de la façon suivante :

1. Lancer le programme sur une collection de fichier
2. Verifier que la sortie est correcte et respecte les spécifications
3. Relancer le programme dans le même contexte qu'en 1, trier la sortie et rediriger le flux de sortie dans un fichier témoin

Pour automatiser les tests nous avons écrit un script bash qui suit les étapes suivantes pour chaque test :

1. Lancer le programme dans le même contexte que lors de la réalisation du fichier témoin, trier la sortie et rediriger la sortie vers un fichier résultat
2. Faire le hash md5 du fichier résultat et du fichier témoin
3. Comparer les deux hash md5
4. Si les hashs diffèrent c'est que le programme ne possède pas les mêmes données en interne et donc le test échoue
5. Si les hashs sont égaux c'est que le programme possède les mêmes données et donc le test réussi

2.2 Critique de la méthode

Avantages : Avec cette méthode de réalisation des tests on s'abstrait de la représentation interne des données. On vérifie uniquement que le programme délivre correctement à l'utilisateur les informations voulues. Ainsi tout changement dans la structure de données interne n'affectera pas les tests tant que le programme délivre les memes informations à l'utilisateur.

De plus la réalisation d'un test supplémentaire se fait facilement et ne prend pas beaucoup de temps par rapport à une analyse compléte de la structure de données interne.

Du fait que nous gardons une sauvegarde du flux de sortie lors d'un état fonctionnel de l'application, le développeur peut comparer visuellement les deux sorties pour trouver plus facilement les éléments divergeants. D'où un gain de temps lors du débuggage

Inconvénients : Avec cette méthode, la formatage des informations à l'utilisateur devient un élément critique. Tout changement dans l'affichage des résultats causera inevitablement l'échec de tous les tests. Toutefois, il peut être facile de créer un script bash permettant de regénérer les fichiers témoins si le développeur est sur des changements qu'il a effectués.

Si un fichier témoin venait à être corrompu cela entrainerait l'échec du test le mettant en jeu.

2.3 Test n° 1

Descriptif : Le test n° 1 réalise le premier exemple donné par le sujet.
Les fichiers analysés sont :

file1.cpp

```cpp
// affiche le message "Hello world"
int main() {
    cout <<"Hello world"<<endl;
    cout<<endl;
    return 0;
}
```

file1.h

```cpp
int main();
```

key1.txt

```
int
world
template
```

Résultat attendu : Nous lançons le programme avec le contexte suivant :

$$tp\_stl \ -e \ -k \ key1.txt \ file1.cpp \ file1.h$$

Nous devons obtenir le résultat ci-dessous après avoir trié la sortie :

file1.res

```
cout        file1.cpp 3 4
endl        file1.cpp 3 4
main        file1.cpp 2       file1.h 1
return        file1.cpp 5
```

2.4 Test n° 2

Descriptif : Le test n° 2 réalise le deuxième exemple donné par le sujet. Les fichiers analysés sont :

file2.cpp

```cpp
// affiche le message "Hello world"
int main() {
    cout <<"Hello world"<<endl;
    cout<<endl;
    return 0;
}
```

file2.h

```cpp
int main();
```

key2.txt

```
int
world
template
```

Résultat attendu : Nous lançons le programme avec le contexte suivant après avoir trié la sortie :

$$tp\_stl \ -k \ key2.txt \ file2.cpp \ file2.h$$

Nous devons obtenir le résultat ci-dessous :

file2.res

```
int         file2.cpp 2         file2.h 1
```

2.5 Test n° 3

Descriptif : Le test n° 3 réalise le test sur le fichier main de notre programme.
Les fichiers analysés sont :

```
//============================================================================
// Name        : Ref_croisee.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
#include <vector>

#include "CmdLine/cmdLine.hpp"
#include "References/Referenceur.hpp"
#include "References/References.hpp"

using namespace std;
using namespace Reference_croisee;

int main( int argc, char** argv )
{/*{{{*/

CmdLine::Arguments args;
CmdLine::Parser parser( "Permet de referencer des mots clefs a travers des fichiers" );
parser.addOption( "exclude,e",  "Inverse le fonctionnement du programme" );
parser.addOption( "keyword,k",  "Specifie la liste des mots clefs a utiliser", true );

try {
    parser.parse( argc, argv, args );

} catch( exception& e ) {
    cout << "Une erreur c'est produit durant la recuperation de la ligne de commande : "
         << endl << e.what() << endl;
}

//------------------------------------------------------------------------
//  On charge les fichiers a referencer
//------------------------------------------------------------------------
vector<string> ficsReferencer;

if( args.count( "__args__" ) ) {
    ficsReferencer = args.get<vector<string> >( "__args__" );

```

```
43  } else {
44      cerr << "Aucun fichier a referencer !" << endl;
45      return 1;
46  }
47
48  //-------------------------------------------------------------------------
49  //  On charge les mots clefs si ils sont fournis
50  //-------------------------------------------------------------------------
51  string fichierMotClef;
52
53  if( args.count( "keyword" ) ) {
54      fichierMotClef = args.get<string>( "keyword" );
55  }
56
57  //-------------------------------------------------------------------------
58  //  L'etat dans lequel mettre le programme
59  //-------------------------------------------------------------------------
60  bool mode( args.count( "exclude" ) );
61
62
63  References refs;
64
65  //-------------------------------------------------------------------------
66  //  On effectue la reference croisee
67  //-------------------------------------------------------------------------
68  try {
69      Referenceur referenceur( fichierMotClef, mode );
70      referenceur.referencer( ficsReferencer, refs );
71
72  } catch( exception& e ) {
73      cerr << "Une erreur est survenue durant la reference croisee : " << endl;
74      cerr << e.what() << endl;
75  }
76
77
78  //-------------------------------------------------------------------------
79  //  On affiche les resultats
80  //-------------------------------------------------------------------------
81  refs.display( cout );
82
83  return 0;
84     }/*}}}*/
```

Résultat attendu : Nous lançons le programme avec le contexte suivant :

$$tp\_stl\ \ file3.cpp$$

Nous devons obtenir le résultat ci-dessous après avoir trié la sortie :

file3.res

```
bool         file3.cpp 60
catch         file3.cpp 30 72
char         file3.cpp 19
cout         file3.cpp 31 81
else         file3.cpp 43
if         file3.cpp 40 53
int         file3.cpp 19 19
namespace         file3.cpp 16 17
return         file3.cpp 45 83
true         file3.cpp 25
try         file3.cpp 27 68
using         file3.cpp 16 17
```

2.6 Test n° 4

Descriptif : Le fichier à analyser est le même que dans le test précédent, seul le contexte d'execution change.

Résultat attendu : Nous lançons le programme avec le contexte suivant :

$$tp\_stl\ file4.cpp$$

Nous devons obtenir le résultat ci-dessous après avoir trié la sortie :

file4.res

```
addOption        file4.cpp 24 25
argc        file4.cpp 19 28
args        file4.cpp 22 28 40 41 53 54 60
Arguments        file4.cpp 22
argv        file4.cpp 19 28
cerr        file4.cpp 44 73 74
CmdLine        file4.cpp 22 23
count        file4.cpp 40 53 60
display        file4.cpp 81
e        file4.cpp 30 32 72 74
endl        file4.cpp 31 32 44 73 74
exception        file4.cpp 30 72
fichierMotClef        file4.cpp 51 54 69
ficsReferencer        file4.cpp 38 41 70
get        file4.cpp 41 54
main        file4.cpp 19
mode        file4.cpp 60 69
parse        file4.cpp 28
Parser        file4.cpp 23
parser        file4.cpp 23 24 25 28
Reference_croisee        file4.cpp 17
referencer        file4.cpp 70
References        file4.cpp 63
Referenceur        file4.cpp 69
referenceur        file4.cpp 69 70
refs        file4.cpp 63 70 81
std        file4.cpp 16
string        file4.cpp 38 41 51 54
vector        file4.cpp 38 41
what        file4.cpp 32 74
```

3 Architecture générale

3.1 Diagramme de classe

3.2 Commentaires

4 Algorithmes principaux

4.1 Parseur pour la ligne de commandes

4.2 Parseur pour les fichiers C++

5 Analyse critique des structures de données

5.1 Structure des identificateurs

5.2 Structure des occurences