Références croisées
Document de réalisation

Tuteurs :
Nabila Benharkat
Eric Guerin

2011

Insa de Lyon
Dpt. Informatique

Table des matières

1 Structure de données de la STL

1.1 Choix pour les identificateurs

Les tables de hashages n'existant pas dans le standard C++03 nous avons décidé d'utiliser la classe **unordered_map** présente dans le standard C++ (C++1x). Cette classe est présente dans le namespace tr1 (ToRealease) et implémente les tables de hashages pour les classes standards de la STL (comme string).

$$tr1 :: std :: unordered\_map < std :: string, \{Occurences\} >$$

1.2 Choix pour les références d'occurences

Nous avons décidé d'utiliser une **map** de **list** pour stocker les occurences des mots clefs. La clef de la structure map est le nom du fichier où apparait l'occurence et la valeur est une liste d'entier indiquant le numéro des lignes où l'on trouve le mot clef.

$$std :: map < std :: string, list < int >>$$

2 Réalisation des tests

2.1 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
```

2.2 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
```

2.3 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  cout << refs;
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
```

2.4 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 Sources de l'application

3.1 Module CmdLine

Parser.hpp

```
// NO_FORMAT ==============================================================================
//
//       Filename:  commandLineParser.hpp
//
//    Description:  Classe permettant d'extraire les informations de la ligne de commande
//        Created:  06/11/2011 00:50:26
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ========================================================================================


#ifndef CmdLineParser_HPP
#define CmdLineParser_HPP

#include    <iostream>
#include    <string>
#include    <map>
#include    <list>
#include    <utility>

namespace CmdLine {

using namespace CmdLine;
class Arguments;


/* ========================================================================================
 *        Class:  Parser
 *  Description:  Permet d'extraire et de decouper les arguments de la ligne de commande
 * ========================================================================================*/
class Parser {

    public:
/*-----------------------------------------------------------------------------
 *  Constructeur
 *-----------------------------------------------------------------------------*/
          Parser( std::string = "" );

/*-----------------------------------------------------------------------------
```

```
42   *  Methodes Publiques
43   *-----------------------------------------------------------------------------*/
44
45           /* ===  FUNCTION  ======================================================
46            *         Name:  addDescription
47            *  Description:  Ajoute une description au programme
48            * ==========================================================================
49           void addDescription( std::string );
50
51           /* ===  FUNCTION  ======================================================
52            *         Name:  addOption
53            *  Description:  Ajoute une option que doit gerer le programme
54            * ==========================================================================
55           void addOption( std::string optionName, std::string description, bool hasArgument
56
57           /* ===  FUNCTION  ======================================================
58            *         Name:  parse
59            *  Description:  Extrait et stocke les informations de la ligne de commande
60            * ==========================================================================
61           void parse( int argc, char** argv, Arguments& args );
62
63           /* ===  FUNCTION  ======================================================
64            *         Name:  display
65            *  Description:  Affiche la description ainsi que les arguments accepte par le pr
66            *                dans le flux specifie
67            * ==========================================================================
68           void display( std::ostream& flux ) const;
69
70
71       private:
72   /*-----------------------------------------------------------------------------
73    *  Les etats que peut prendre l'objet
74    *-----------------------------------------------------------------------------*/
75           enum State { Free, ShortCondition, LongCondition, FreeArgument };
76           State _state;
77
78
79   /*-----------------------------------------------------------------------------
80    *  Les structures de donnees pour stocker les arguments
81    *-----------------------------------------------------------------------------*/
82           struct optValue;
83           typedef std::pair<const std::string, optValue*> lItem;
84           typedef std::pair<const char, optValue*> sItem;
85
86           struct optValue {
87                std::string description;
```

```
88                bool hasArgument;
89                lItem* lOption;
90                sItem* sOption;
91           };
92
93           std::map< const std::string, optValue* > _lOptions;
94           std::map< const char, optValue* > _sOptions;
95           std::list<optValue> _options; // pas de vector car la zone memoire bouge
96           std::string _description;
97           std::string _freeArgs;
98
99   /*-----------------------------------------------------------------------------
100   *  Methodes privees
101   *-----------------------------------------------------------------------------*/
102
103          /* ===  FUNCTION  ======================================================
104           *         Name:  processInput
105           *  Description:  Traite la ligne de commande
106           * ======================================================================*/
107          void processInput( std::string::iterator& it, Arguments& args );
108
109          /* ===  FUNCTION  ======================================================
110           *         Name:  ChangeState
111           *  Description:  Determine l'etat de l'objet
112           * ======================================================================*/
113          void changeState( std::string::iterator& it );
114
115
116
117
118  };
119
120  //Surcharge de l'operateur de flux
121  std::ostream& operator<<( std::ostream& flux,  const Parser& parser );
122
123  }/*}}}*/
124  #endif
```

Parser.cpp

```
1  // ============================================================================================
2  //
3  //       Filename:  Parser.cpp
4  //
5  //    Description:  Permet d'extraire les informations de la ligne de commande
6  //        Created:  06/11/2011 01:43:35
7  //       Compiler:  g++
8  //
9  //         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // ============================================================================================
12
13 #include    "Parser.hpp"
14 #include    "Arguments.hpp"
15 #include    <string>
16 #include    <map>
17 #include    <utility>
18
19 namespace CmdLine {
20
21 using namespace CmdLine;
22 using namespace std;
23
24 /*-------------------------------------------------------------------------------
25  *  Constructeurs
26  *-------------------------------------------------------------------------------*/
27 Parser::Parser( string description ):
28      _state( Free ), _description( description )
29 {}
30
31
32 /*-------------------------------------------------------------------------------
33  *  Methode Publiques
34  *-------------------------------------------------------------------------------*/
35 void Parser::addDescription( string description )
36 {/*{{{*/
37      _description = description;
38 }/*}}}*/
39
40 void Parser::addOption( string optionName, string description, bool hasArgument )
41 {/*{{{*/
42
43      string lOption, sOption;
44
```

```
45        size_t pos = optionName.find_first_of( ',' );
46
47        lOption = optionName.substr( 0,  pos );
48        sOption = optionName.substr( ( pos == string::npos ) ? optionName.size() : ++pos );
49
50        lItem* lIt = &( *( _lOptions.insert( make_pair( lOption, ( optValue* )0 ) ).first ) );
51        sItem* sIt = 0;
52
53        if( !sOption.empty() ) {
54             sIt = &( *( _sOptions.insert( make_pair( sOption.at( 0 ), ( optValue* )0 ) ).first
55        }
56
57        optValue value = { description, hasArgument, lIt, sIt };
58        _options.push_back( value );
59
60        lIt->second = &( _options.back() );
61
62        if ( sIt ) {
63             sIt->second = lIt->second;
64        }
65   }/*}}}*/
66
67   void Parser::display( ostream& flux ) const
68   {/*{{{*/
69
70        string option;
71        flux << _description << endl << endl;
72
73        flux << "Liste des arguments : " << endl;
74
75        for( list<optValue>::const_iterator it = _options.begin(); it != _options.end(); it++ )
76
77             flux << "\t";
78
79             if( it->sOption ) {
80                  option += "-";
81                  option += it->sOption->first;
82                  option += ", ";
83             }
84
85             option += "--";
86             flux.width( 18 );
87             flux << left << option + it->lOption->first;
88
89             flux.width( 5 );
90
```

```
91            if ( it->hasArgument ) {
92                 flux << left << "arg";
93
94            } else {
95                 flux << "";
96            }
97
98            flux << it->description << endl;
99
100           option.clear();
101      }
102
103 }/*}}}*/
104
105 void Parser::parse( int argc, char** argv, Arguments& args )
106 {/*{{{*/
107
108      string cmdLine;
109
110      for ( char** it = argv + 1; it < argv + argc; it++ ) {
111           cmdLine += *it;
112           cmdLine += " ";
113      }
114
115      cmdLine += "  ";
116
117      string::iterator it = cmdLine.begin();
118
119      while( it != cmdLine.end() ) {
120
121           changeState( it );
122           processInput( it, args );
123      }
124
125      if( !_freeArgs.empty() ) {
126          args._arguments.insert( make_pair( "__args__", _freeArgs ) );
127      }
128      //    for( map<string, string>::iterator it = args._arguments.begin();
129      //         it != args._arguments.end(); it++ ) {
130      //        cout << it->first << " : " << it->second << endl;
131      //    }
132
133
134 }/*}}}*/
135
136
```

```
137 /*--------------------------------------------------------------------------------
138  *  Methodes Privees
139  *-------------------------------------------------------------------------------*/
140 void Parser::changeState( string::iterator& it )
141 {/*{{{*/
142 
143     if( *it == '-' ) {
144         _state = ( _state == ShortCondition ) ? LongCondition : ShortCondition;
145 
146     } else if( *it == ' ' ) {
147         _state = Free;
148 
149     } else if( _state == ShortCondition ) {
150         _state = ShortCondition;
151 
152     } else {
153         _state = FreeArgument;
154     }
155 
156 }/*}}}*/
157 
158 void Parser::processInput( string::iterator& it, Arguments& args )
159 {/*{{{*/
160 
161 
162     if( _state == Free ) {
163         it++;
164 
165     } else if( _state == FreeArgument ) {
166 
167         while( *it != ' ' ) {
168             _freeArgs += *it;
169             it++;
170         }
171 
172         _freeArgs += ',';
173 
174     } else if ( _state == LongCondition ) {
175         string key;
176         it++;
177 
178         while( *it != '=' && *it != ' ' ) {
179             key += *it;
180             it++;
181         }
182 
```

```
183             if( !_lOptions.count( key ) ) {
184                  throw UnknownOption( key );
185             }
186
187             lItem& item = *( _lOptions.find( key ) );
188
189             if( ( item.second->hasArgument && *it == ' ' )
190                 || ( !item.second->hasArgument && *it == '=' ) ) {
191                  throw ArgumentBadFormatted();
192             }
193
194             string value;
195
196             if( !item.second->hasArgument ) {
197                  value = "42";
198
199             } else {
200                  it++;
201
202                  while( *it != ' ' ) {
203                       value += *it;
204                       it++;
205                  }
206             }
207
208             if( value.empty() ) {
209                  throw ArgumentBadFormatted();
210             }
211
212             args._arguments.insert( make_pair( key, value ) );
213
214        } else if ( _state == ShortCondition ) {
215
216             if ( *it == '-' ) {
217                  it++;
218
219             } else {
220
221                  string key;
222                  key += *it;
223
224                  if ( !_sOptions.count( *it ) ) {
225                       throw UnknownOption( key );
226                  }
227
228                  sItem& item = *( _sOptions.find( *it ) );
```

```
229                 key = item.second->lOption->first;
230
231                 it++;
232
233                 if ( !item.second->hasArgument ) {
234                     args._arguments.insert( make_pair( key, "42" ) );
235
236                 } else {
237
238                     if ( *it != ' ' ) {
239                         throw ArgumentBadFormatted();
240                     }
241
242                     ++it;
243                     string value;
244
245                     while( *it != ' ' ) {
246                         value += *it;
247                         it++;
248                     }
249
250                     if( value.empty() ) {
251                         throw  ArgumentBadFormatted();
252                     }
253
254                     args._arguments.insert( make_pair( key, value ) );
255                 }
256
257             }
258         }
259
260
261
262 }/*}}}*/
263
264
265 /*-------------------------------------------------------------------------
266  *  Surcharge Operateur
267  *-------------------------------------------------------------------------*/
268 ostream& operator<<( ostream& flux, const Parser& parser )
269 {/*{{{*/
270
271     parser.display( flux );
272
273     return flux;
274 }/*}}}*/
```

```
275
276 }/*}}}*/
```

Arguments.hpp

```
// NO_FORMAT ==============================================================================
//
//       Filename:  cmdLineArgument.hpp
//
//    Description:  Permet de stocker les arguments passe en parametre du programme
//        Created:  06/11/2011 21:35:00
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ========================================================================================


#ifndef CmdLineArgument_HPP
#define CmdLineArgument_HPP

#include    <string>
#include    <map>

#include        "Exceptions.hpp"
#include    "StringTo.hpp"

namespace CmdLine {

using namespace CmdLine;

/* ========================================================================================
 *        Class:  Argument
 *  Description:  Permet de gerer les options de la ligne de commande
 * ========================================================================================*/
class Arguments {

    public:
        /* ===  FUNCTION  ======================================================================
         *         Name:  count
         *  Description:  Permet de savoir si une option est presente
         *                Il faut donner le nom long de l'option
         * ======================================================================================*/
        bool count( std::string arg ) const {
            return _arguments.count( arg );
        }

        /* ===  FUNCTION  ======================================================================
         *         Name:  get
```

```
45              *  Description:  Permet de recuperer une option
46              * =====================================================================================
47              template<typename T>
48              T get( const char* arg ) {
49
50                   if ( !_arguments.count( arg ) )
51                        {   throw InvalidKey(); }
52
53                   return stringTo<T>( _arguments[arg] );
54              }
55
56              /* ===  FUNCTION  ======================================================================
57               *         Name:  get
58               *  Description:  Permet de recuperer une option
59               * =====================================================================================
60              template<typename T>
61              T get( std::string& arg ) {
62
63                   if ( !_arguments.count( arg ) )
64                        {   throw InvalidKey(); }
65
66                   return stringTo<T>( _arguments[arg] );
67              }
68
69
70         private:
71              /*-----------------------------------------------------------------------------
72               *  Structures de donnees
73               *-----------------------------------------------------------------------------*/
74              std::map<std::string, std::string> _arguments;
75
76              /*-----------------------------------------------------------------------------
77               *  Classe friend
78               *-----------------------------------------------------------------------------*/
79              friend class Parser;
80   };
81
82   }/*}}}*/
83
84   #endif
```

StringTo.hpp

```
// ============================================================================================
//
//       Filename:  StringTo.hpp
//
//    Description:  Templates pour convertir les chaines de caracteres
//        Created:  09/11/2011 01:00:03
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ============================================================================================

#ifndef CmdLineStringTo_HPP
#define CmdLineStringTo_HPP

#include        <string>
#include        <vector>
#include        <sstream>
#include        "Exceptions.hpp"


namespace CmdLine {

using namespace CmdLine;

#define STATIC_ASSERT( x , MSG ) typedef char __STATIC_ASSERT__##MSG[( x )?1:-1]

    template<typename T>
        T stringTo( const std::string& arg )
    {/*{{{*/

         STATIC_ASSERT( sizeof(T) != sizeof(T),  Impossible_de_convertir_l_argument_vers_ce_
         return T();
    }/*}}}*/

/*-----------------------------------------------------------------------------
 *  Types simples
 *-----------------------------------------------------------------------------*/
    template<>
        inline int stringTo<int>( const std::string& arg )
    {/*{{{*/
         std::stringstream s( arg );
         int value = 0;

```

```
45          s >> value;
46
47          if ( s.fail() )
48               { throw BadConvertion(); }
49
50          return value;
51      }/*}}}*/
52
53      template<>
54          inline float stringTo<float>( const std::string& arg )
55      {/*{{{*/
56          std::stringstream s( arg );
57          float value = 0;
58
59          s >> value;
60
61          if ( s.fail() )
62               { throw BadConvertion(); }
63
64          return value;
65      }/*}}}*/
66
67      template<>
68          inline double stringTo<double>( const std::string& arg )
69      {/*{{{*/
70          std::stringstream s( arg );
71          double value = 0;
72
73          s >> value;
74
75          if ( s.fail() )
76               { throw BadConvertion(); }
77
78          return value;
79      }/*}}}*/
80
81      template<>
82          inline bool stringTo<bool>( const std::string& arg )
83      {/*{{{*/
84          std::stringstream s( arg );
85          int value = 0;
86
87          s >> value;
88
89          if ( s.fail() )
90               { throw BadConvertion(); }
```

```
91
92          return ( value );
93      }/*}}}*/
94
95      template<>
96          inline std::string stringTo<std::string>( const std::string& arg )
97      {/*{{{*/
98          return arg;
99      }/*}}}*/
100
101
102 /*-------------------------------------------------------------------------------
103  *  Types composes
104  *-------------------------------------------------------------------------------*/
105     template<>
106         inline std::vector<std::string> stringTo<std::vector< std::string> >( const std::str
107     {/*{{{*/
108
109         std::string arg = cpArg;
110         std::vector<std::string> conteneur;
111         const char separator = ',';
112
113         int found = arg.find_first_of( separator );
114
115         while( found != ( ( int ) std::string::npos ) ) {
116             if( found > 0 ) {
117                 conteneur.push_back( arg.substr( 0, found ) );
118             }
119
120             arg = arg.substr( found + 1 );
121             found = arg.find_first_of( separator );
122         }
123
124         if ( arg.length() > 0 ) {
125             conteneur.push_back( arg );
126         }
127
128         return conteneur;
129     }/*}}}*/
130
131     template<>
132         inline std::vector<int> stringTo<std::vector<int> >( const std::string& arg )
133     {/*{{{*/
134
135         std::vector<int> conteneur;
136         const char separator = ',';
```

```
137           std::stringstream buffer;
138           std::string cpArg = arg;
139
140           int found = cpArg.find_first_of( separator );
141           int val = 0;
142
143           while( found != ( ( int ) std::string::npos ) ) {
144
145                if( found > 0 ) {
146                     buffer << cpArg.substr( 0, found );
147                     buffer >> val;
148
149                     if ( buffer.fail() )
150                     { throw BadConvertion(); }
151
152                     conteneur.push_back( val );
153                }
154
155                cpArg = cpArg.substr( found + 1 );
156                found = cpArg.find_first_of( separator );
157                buffer.clear();
158           }
159
160           if ( arg.length() > 0 ) {
161             buffer << cpArg.substr( 0, found );
162             buffer >> val;
163             if ( buffer.fail() )
164             { throw BadConvertion(); }
165             conteneur.push_back( val );
166           }
167
168           return conteneur;
169      }/*}}}*/
170
171      template<>
172          inline std::vector<double> stringTo<std::vector<double> >( const std::string& arg )
173      {/*{{{*/
174
175           std::vector<double> conteneur;
176           const char separator = ',';
177           std::stringstream buffer;
178           std::string cpArg = arg;
179
180           int found = cpArg.find_first_of( separator );
181           double val = 0;
182
```

```
183          while( found != ( ( int ) std::string::npos ) ) {
184
185               if( found > 0 ) {
186                    buffer << cpArg.substr( 0, found );
187                    buffer >> val;
188
189                    if ( buffer.fail() )
190                    { throw BadConvertion(); }
191
192                    conteneur.push_back( val );
193               }
194
195               cpArg = cpArg.substr( found + 1 );
196               found = cpArg.find_first_of( separator );
197               buffer.clear();
198          }
199
200          if ( arg.length() > 0 ) {
201            buffer << cpArg.substr( 0, found );
202            buffer >> val;
203
204            if ( buffer.fail() )
205            { throw BadConvertion(); }
206
207            conteneur.push_back( val );
208          }
209
210          return conteneur;
211     }/*}}}*/
212
213     template<>
214         inline std::vector<float> stringTo<std::vector<float> >( const std::string& arg )
215     {/*{{{*/
216
217          std::vector<float> conteneur;
218          const char separator = ',';
219          std::stringstream buffer;
220          std::string cpArg = arg;
221
222          int found = cpArg.find_first_of( separator );
223          double val = 0;
224
225          while( found != ( ( int ) std::string::npos ) ) {
226
227               if( found > 0 ) {
228                    buffer << cpArg.substr( 0, found );
```

```
229                     buffer >> val;
230
231                     if ( buffer.fail() )
232                     { throw BadConvertion(); }
233
234                     conteneur.push_back( val );
235                 }
236
237                 cpArg = cpArg.substr( found + 1 );
238                 found = cpArg.find_first_of( separator );
239                 buffer.clear();
240             }
241
242             if ( arg.length() > 0 ) {
243               buffer << cpArg.substr( 0, found );
244               buffer >> val;
245
246               if ( buffer.fail() )
247               { throw BadConvertion(); }
248
249               conteneur.push_back( val );
250             }
251
252             return conteneur;
253         }/*}}}*/
254 }
255
256 #endif
```

Exceptions.hpp

```
// NO_FORMAT ==============================================================================
//
//       Filename:  Exceptions.hpp
//
//    Description:  Les exceptions pouvant etre lancees
//        Created:  09/11/2011 01:07:40
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ========================================================================================

#ifndef CmdLineExceptions_HPP
#define CmdLineExceptions_HPP

#include        <exception>

/*-----------------------------------------------------------------------------
 *  Exceptions
 *-----------------------------------------------------------------------------*/

namespace CmdLine {

using namespace CmdLine;


    /* ========================================================================================
     *        Class:  UnknownCnvertion
     *  Description:  Exception lance si on ne sait pas convertir la chaine vers le format
     * ========================================================================================*/
    class UnknownConvertion : public std::exception {
         public:
              virtual const char* what() const throw()
              { return "Don't know how to convert the argument to this type !"; }
    };


    /* ========================================================================================
     *        Class:  BadConvertion
     *  Description:  Exception lance si la convertion a echoue
     * ========================================================================================*/
    class BadConvertion : public std::exception {
         public:
              virtual const char* what() const throw()
```

```
45                { return "The convertion of the argument to this type failed !"; }
46      };
47
48
49      /* ================================================================================
50       *        Class:  InvalidKey
51       *  Description:  Exception lance si l'option ne correspond a aucune clef
52       * ================================================================================
53      class InvalidKey : public std::exception {
54           public:
55                virtual const char* what() const throw()
56                { return "The key specified don't exist ! Check first with count()"; }
57      };
58
59
60      /* ================================================================================
61       *        Class:  ArgumentBadFormatted
62       *  Description:  Exception lance lorsque la ligne de commande n'est pas bien formate
63       * ================================================================================
64      class ArgumentBadFormatted : public std::exception {
65           public:
66                const char* what() const throw()
67                { return "Arguments are not well formatted !"; }
68      };
69
70
71      /* ================================================================================
72       *        Class:  UnknownOption
73       *  Description:  Exception lance lorsqu'une option n'existe pas
74       * ================================================================================
75      class UnknownOption : public std::exception {
76           public:
77                UnknownOption( std::string& option ): _option( option ) {}
78                virtual ~UnknownOption() throw() {}
79
80                const char* what() const throw()
81                { return std::string( "Option not recognized \"" + _option + "\"" ).c_str(); }
82
83           private:
84                std::string _option;
85      };
86
87  }
88  #endif
```

3.2 Module References

References.hpp

```
// ============================================================================================
//
//       Filename:  References.hpp
//
//    Description:  Permet de stocker les resultats des references croisees
//        Created:  18/11/2011 22:29:34
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ============================================================================================


#ifndef References_HPP
#define References_HPP


#include    <map>
#include    <list>
#include    <tr1/unordered_map>

namespace Reference_croisee {

using namespace Reference_croisee;

//-----------------------------------------------------------------------
// Role de la classe References
// Description : Permet de stocker les occurences des references croisees
//
//-----------------------------------------------------------------------
class References {

    public:
//----------------------------------------------------------------------
//  METHODES PUBLIQUES
//----------------------------------------------------------------------

        // ===  FUNCTION  ======================================================================
        //         Name:  add
        //  Description:  Permet d'ajouter une reference croisee au conteneur
        // ====================================================================================
        void add( const std::string& motClef, const std::string& nomFichier, int ligne );

```

```
44          // ===  FUNCTION  ======================================================================
45          //         Name:  display
46          //  Description:  Permet d'afficher toutes les references dans le flux fournit en e
47          // =====================================================================================
48          void display( std::ostream& flux ) const;
49
50
51      protected:
52  //-----------------------------------------------------------------------
53  //  ATTRIBUTS MEMBRES
54  //-----------------------------------------------------------------------
55
56          // Structure interne du conteneur
57          std::tr1::unordered_map<std::string, std::map<std::string, std::list<int> > > _refer
58
59  };
60
61  std::ostream& operator<<( std::ostream& flux,  const References& ref );
62
63  }/*}}}*/
64
65  #endif
```

References.cpp

```cpp
// ============================================================================================
//
//       Filename:  References.cpp
//
//    Description:
//        Created:  18/11/2011 22:50:44
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ============================================================================================



#include <iostream>
#include "References.hpp"

namespace Reference_croisee {

using namespace std;
using namespace Reference_croisee;


//-----------------------------------------------------------------------
//  METHODES PUBLIQUES
//-----------------------------------------------------------------------
void References::add( const string& motClef, const string& nomFichier, const int ligne )
{/*{{{*/


    if( !_references.count( motClef ) ) {
        _references.insert( make_pair( motClef, map<string, list<int> >() ) );
        _references[motClef].insert( make_pair( nomFichier, list<int>( 1, ligne ) ) );

    } else if( !_references[motClef].count( nomFichier ) ) {
        _references[motClef].insert( make_pair( nomFichier, list<int>( 1, ligne ) ) );

    } else {
        _references[motClef][nomFichier].push_back( ligne );
    }


}/*}}}*/

```

```
45  void References::display( ostream& flux ) const
46  {/*{{{*/
47
48      tr1::unordered_map<string, map<string, list<int> > >::const_iterator itClef;
49      map<string, list<int> >::const_iterator itFic;
50      list<int>::const_iterator itLigne;
51
52
53      for( itClef = _references.begin(); itClef != _references.end(); itClef++ ) {
54          //flux.width(15);
55          //flux << left;
56          flux << itClef->first << "\t";
57
58          for( itFic = itClef->second.begin(); itFic != itClef->second.end(); itFic++ ) {
59              flux << itFic->first;
60
61              for( itLigne = itFic->second.begin(); itLigne != itFic->second.end(); itLigne++
62                  flux << ' ' << *itLigne;
63
64              }
65
66              flux << '\t';
67          }
68
69          flux << endl;
70      }
71
72  }/*}}}*/
73
74
75
76  //-------------------------------------------------------------------------
77  //  SURCHARGES OPERATEURS
78  //-------------------------------------------------------------------------
79  ostream& operator<<( ostream& flux,  const References& ref ) {
80
81      ref.display( flux );
82
83      return flux;
84
85  }
86
87  }
```

Referenceur.hpp

```
// ============================================================================================
//
//       Filename:  References.hpp
//
//    Description:  Interface de la classe References
//                  Permet de visualiser la repartition de mots clefs dans une
//                  collection de fichiers
//        Created:  15/11/2011 23:30:30
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ============================================================================================


#ifndef Referenceur_HPP
#define Referenceur_HPP


#include    <string>
#include    <vector>
#include    <tr1/unordered_set>
#include    "FichierLu.hpp"
#include    "References.hpp"

namespace Reference_croisee {

using namespace Reference_croisee;

//-----------------------------------------------------------------------------
// Role de la classe Referenceur
// Description : Permet d'extraire les mots clefs dans une
//               collection de fichiers
//-----------------------------------------------------------------------------
class Referenceur {

    public:
//-----------------------------------------------------------------------------
//  CONSTRUCTEURS
//-----------------------------------------------------------------------------
        Referenceur( const std::string fichierMotClef = std::string(),
                     const bool modeInverse = false );

//-----------------------------------------------------------------------------
```

```
45  //  METHODES PUBLIQUES
46  //------------------------------------------------------------------------
47
48          // ===  FUNCTION  ======================================================================
49          //         Name:  chargerMotsClefs
50          //  Description:  Permet de charger des mots clefs a partir d'un fichier
51          // =====================================================================================
52          void chargerMotsClefs( const std::string& nomFichier );
53
54          // ===  FUNCTION  ======================================================================
55          //         Name:  chargerMotsClefsCpp
56          //  Description:  Permet de definir les mots clefs C++ standard comme des mots clefs
57          // =====================================================================================
58          void chargerMotsClefsCpp();
59
60          // ===  FUNCTION  ======================================================================
61          //         Name:  setModeInverse
62          //  Description:  Permet de passer le parseur en mode inverse
63          // =====================================================================================
64          inline void setModeInverse( const bool mode );
65
66          // ===  FUNCTION  ======================================================================
67          //         Name:  referencer
68          //  Description:  Permet de chercher les mots clefs une collection de fichier
69          //                Les resultats sont stockes dans refs
70          // =====================================================================================
71          void referencer( const std::vector<std::string>& fic, References& refs );
72
73      protected:
74          enum Mode { Normal, Inverse };          // Les differents mode du parseur
75          enum Etat { Separateur, Commentaire, MotClef, Preprocesseur, Literal }; // Les etats
76
77
78
79  //------------------------------------------------------------------------
80  //  ATTRIBUTS MEMBRES
81  //------------------------------------------------------------------------
82          Mode _mode;
83          Etat _etat;
84
85          std::tr1::unordered_set<std::string> _motsClefs; // le conteneur des mots clefs
86
87
88
89  //------------------------------------------------------------------------
90  //  METHODES PROTEGES
```

```
91  //-------------------------------------------------------------------------
92
93          // ===  FUNCTION  ======================================================================
94          //         Name:  estInserable
95          //  Description:  retourne vrai si l'identificateur est un mot clef
96          // =====================================================================================
97          inline bool estInserable( const std::string& mot ) const;
98
99          // ===  FUNCTION  ======================================================================
100         //         Name:  isSeparateur
101         //  Description:  Retourne vrai si le caractere est un separateur
102         // =====================================================================================
103         bool isSeparateur( const char c ) const;
104
105         // ===  FUNCTION  ======================================================================
106         //         Name:  changerEtat
107         //  Description:  Definit le nouvel etat de l'automate
108         // =====================================================================================
109         void changerEtat( FichierLu& fic );
110
111         // ===  FUNCTION  ======================================================================
112         //         Name:  lireFlux
113         //  Description:  Avance dans le flux de donnees en fonction de l'etat de l'automate
114         // =====================================================================================
115         void lireFlux( FichierLu& fic, References& refs );
116
117
118
119 //-------------------------------------------------------------------------
120 //  METHODES ETATS
121 //-------------------------------------------------------------------------
122
123         // ===  FUNCTION  ======================================================================
124         //         Name:  lirePreprocesseur
125         //  Description:  Traite les instructions preprocesseurs
126         // =====================================================================================
127         void lirePreprocesseur( FichierLu& fic, References& refs );
128
129         // ===  FUNCTION  ======================================================================
130         //         Name:  lireSeparateur
131         //  Description:  Traite les separateurs
132         // =====================================================================================
133         void lireSeparateur( FichierLu& fic, References& refs );
134
135         // ===  FUNCTION  ======================================================================
136         //         Name:  lireIdentificateur
```

```
137         //  Description:  Traite les identificateurs
138         // ======================================================================
139         void lireIdentificateur( FichierLu& fic, References& refs );
140 
141         // ===  FUNCTION  ======================================================
142         //         Name:  lireCommentaire
143         //  Description:  Traite les commentaires
144         // ======================================================================
145         void lireCommentaire( FichierLu& fic, References& refs );
146 
147         // ===  FUNCTION  ======================================================
148         //         Name:  lireLiteral
149         //  Description:  Traire les chaines de caracteres
150         // ======================================================================
151         void lireLiteral( FichierLu& fic, References& refs );
152 
153 
154 };
155 
156 }/*}}}*/
157 #endif
```

Referenceur.cpp

```
// ============================================================================================
//
//       Filename:  Referenceur.cpp
//
//    Description:  Implementation de la classe Referenceur
//                  Permet de visualiser l'aparition de mots clefs dans une collection
//                  de fichiers
//        Created:  15/11/2011 23:37:07
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ============================================================================================

using namespace std;

#include    <iostream>
#include    <limits>

#include    "Referenceur.hpp"

namespace Reference_croisee {

using namespace Reference_croisee;



//--------------------------------------------------------------------------
//  CONSTRUCTEURS
//--------------------------------------------------------------------------
Referenceur::Referenceur ( const string fichierMotClef, const bool modeInverse ):
    _mode( modeInverse ? Inverse : Normal ), _etat( Separateur )
{/*{{{*/

#ifdef MAP
    cout << "Appel au constructeur de <Referenceur>" << endl;
#endif

    chargerMotsClefsCpp();

    if( !fichierMotClef.empty() ) {
        chargerMotsClefs( fichierMotClef );
    }

```

```
45  }/*}}}*/
46
47  //-------------------------------------------------------------------------
48  //  METHODES PUBLIQUES
49  //-------------------------------------------------------------------------
50  void Referenceur::chargerMotsClefs( const string& nomFichier )
51  {/*{{{*/
52
53
54      ifstream fichierMotClef;
55
56      //---------------------------------------------------------------------
57      //  Si le fichier ne peut etre ouvert ou si la lecture echoue
58      //  une exception sera lance
59      //---------------------------------------------------------------------
60      fichierMotClef.exceptions( ifstream::failbit );
61      fichierMotClef.open( nomFichier.c_str(), ios::in );
62      fichierMotClef.exceptions( ifstream::badbit );
63
64      //---------------------------------------------------------------------
65      //  On extrait la liste de mot clef
66      //---------------------------------------------------------------------
67      _motsClefs.clear();
68      string motRecupere;
69
70      while ( fichierMotClef >> motRecupere ) {
71          _motsClefs.insert( motRecupere );
72          fichierMotClef.ignore( numeric_limits<int>::max(), '\n' );
73          // ignore le nombre de caractere "valeur max d'un entier" jusqu'a rencontrer \n
74      }
75
76      fichierMotClef.close( );
77
78  }/*}}}*/
79
80  void Referenceur::chargerMotsClefsCpp()
81  {/*{{{*/
82
83      _motsClefs.clear();
84      _motsClefs.insert( "asm" );
85      _motsClefs.insert( "auto" );
86      _motsClefs.insert( "break" );
87      _motsClefs.insert( "bool" );
88      _motsClefs.insert( "case" );
89      _motsClefs.insert( "catch" );
90      _motsClefs.insert( "cout" );
```

```
91      _motsClefs.insert( "char" );
92      _motsClefs.insert( "class" );
93      _motsClefs.insert( "const" );
94      _motsClefs.insert( "const_cast" );
95      _motsClefs.insert( "continue" );
96      _motsClefs.insert( "default" );
97      _motsClefs.insert( "delete" );
98      _motsClefs.insert( "do" );
99      _motsClefs.insert( "double" );
100     _motsClefs.insert( "dynamic_cast" );
101     _motsClefs.insert( "else" );
102     _motsClefs.insert( "enum" );
103     _motsClefs.insert( "extern" );
104     _motsClefs.insert( "export" );
105     _motsClefs.insert( "explicit" );
106     _motsClefs.insert( "false" );
107     _motsClefs.insert( "float" );
108     _motsClefs.insert( "for" );
109     _motsClefs.insert( "friend" );
110     _motsClefs.insert( "goto" );
111     _motsClefs.insert( "if" );
112     _motsClefs.insert( "inline" );
113     _motsClefs.insert( "int" );
114     _motsClefs.insert( "long" );
115     _motsClefs.insert( "mutable" );
116     _motsClefs.insert( "namespace" );
117     _motsClefs.insert( "new" );
118     _motsClefs.insert( "operator" );
119     _motsClefs.insert( "private" );
120     _motsClefs.insert( "protected" );
121     _motsClefs.insert( "public" );
122     _motsClefs.insert( "register" );
123     _motsClefs.insert( "reinterpret_cast" );
124     _motsClefs.insert( "return" );
125     _motsClefs.insert( "short" );
126     _motsClefs.insert( "signed" );
127     _motsClefs.insert( "sizeof" );
128     _motsClefs.insert( "static" );
129     _motsClefs.insert( "static_cast" );
130     _motsClefs.insert( "struct" );
131     _motsClefs.insert( "switch" );
132     _motsClefs.insert( "template" );
133     _motsClefs.insert( "this" );
134     _motsClefs.insert( "throw" );
135     _motsClefs.insert( "try" );
136     _motsClefs.insert( "true" );
```

```
137     _motsClefs.insert( "typedef" );
138     _motsClefs.insert( "typeid" );
139     _motsClefs.insert( "typename" );
140     _motsClefs.insert( "unsigned" );
141     _motsClefs.insert( "union" );
142     _motsClefs.insert( "using" );
143     _motsClefs.insert( "virtual" );
144     _motsClefs.insert( "void" );
145     _motsClefs.insert( "volatile" );
146     _motsClefs.insert( "while" );
147     _motsClefs.insert( "wchar_t" );
148 }/*}}}*/
149
150 void Referenceur::referencer( const vector<string>& fichiers, References& refs )
151 {/*{{{*/
152
153     vector<string>::const_iterator it;
154     FichierLu fichier;
155
156     for( it = fichiers.begin(); it != fichiers.end(); it++ ) {
157
158         fichier.open( it->c_str() );
159
160         while( !fichier.eof() ) {
161
162             changerEtat( fichier );
163             lireFlux( fichier, refs );
164
165         }
166
167         fichier.close();
168     }
169
170 }/*}}}*/
171
172 inline void Referenceur::setModeInverse( const bool mode )
173 {/*{{{*/
174     _mode = ( mode ) ? Inverse : Normal;
175 }/*}}}*/
176
177
178
179 //-------------------------------------------------------------------------
180 //  METHODES PROTEGES
181 //-------------------------------------------------------------------------
182 inline bool Referenceur::estInserable( const string& mot ) const
```

```
183 {/*{{{*/
184
185     const char c = mot.at( 0 );
186
187     if( c >= '0' && c <= '9' ) {
188         return false;
189     }
190
191     return  ( _mode == Normal ) ? _motsClefs.count( mot ) :
192             !_motsClefs.count( mot );
193
194
195 }/*}}}*/
196
197 inline bool Referenceur::isSeparateur( const char c ) const
198 {/*{{{*/
199
200     // Je me suis base sur la table ASCII
201     return ( c >= -1 && c <= '/' ) ||
202            ( c >= ':' && c <= '@' ) ||
203            ( c >= '[' && c <= '^' ) ||
204            ( c >= '{' && c <= '~' ) ||
205            ( c == '`' );
206
207 }/*}}}*/
208
209 void Referenceur::changerEtat( FichierLu& fic )
210 {/*{{{*/
211
212     const char c = fic.peek();
213
214     if( c == '#' ) {
215         _etat = Preprocesseur;
216
217     } else if(  c == '/' ) {
218         _etat = Commentaire;
219
220     } else if(  c == '"' || c == '\'' ) {
221         _etat = Literal;
222
223     } else if( isSeparateur( c ) ) {
224         _etat = Separateur;
225
226     } else {
227         _etat = MotClef;
228     }
```

```
229 }/*}}}*/
230
231 void Referenceur::lireFlux( FichierLu& fic, References& refs )
232 {/*{{{*/
233
234     switch( _etat ) {
235
236         case Preprocesseur:
237             lirePreprocesseur( fic, refs );
238             break;
239
240         case Separateur:
241             lireSeparateur( fic, refs );
242             break;
243
244         case Commentaire:
245             lireCommentaire( fic, refs );
246             break;
247
248         case MotClef:
249             lireIdentificateur( fic, refs );
250             break;
251
252         case Literal:
253             lireLiteral( fic, refs );
254             break;
255     }
256
257 }/*}}}*/
258
259
260
261
262 //-----------------------------------------------------------------------------
263 //  METHODES ETATS
264 //-----------------------------------------------------------------------------
265 void Referenceur::lirePreprocesseur( FichierLu& fic, References& refs )
266 {/*{{{*/
267
268     char last = fic.get();
269
270     while( fic.peek() != '\n' || last == '\\' ) {
271         last = fic.get();
272     }
273
274     fic.get();
```

```
275
276
277 }/*}}}*/
278
279 void Referenceur::lireCommentaire( FichierLu& fic, References& refs )
280 {/*{{{*/
281
282     fic.get();
283
284     if( fic.peek() == '/' ) {
285         while ( !fic.eof() && fic.get() != '\n' );
286
287     } else if( fic.peek() == '*' ) {
288         while ( !fic.eof() && ( fic.get() != '*' || fic.peek() != '/' ) );
289
290         fic.get();
291     }
292
293 }/*}}}*/
294
295 void Referenceur::lireIdentificateur( FichierLu& fic, References& refs )
296 {/*{{{*/
297
298     string mot;
299     mot.append( 1, fic.get() );
300
301     while( !fic.eof() && !isSeparateur( fic.peek() ) ) {
302         mot.append( 1, fic.get() );
303     }
304
305     if( estInserable( mot ) ) {
306         refs.add( mot, fic.getNomFichier(), fic.getNbLignesLues() );
307     }
308
309 }/*}}}*/
310
311 void Referenceur::lireSeparateur( FichierLu& fic, References& refs )
312 {/*{{{*/
313
314     fic.get();
315 }/*}}}*/
316
317 void Referenceur::lireLiteral( FichierLu& fic, References& refs )
318 {/*{{{*/
319
320     char last = fic.get();
```

```
321
322     if( last == '"' ) {
323         while( fic.peek() != '"' || last == '\\' ) {
324             last = fic.get();
325         }
326
327         fic.get();
328
329     } else if( last == '\'' ) {
330         while( fic.peek() != '\'' || last == '\\' ) {
331             last = fic.get();
332         }
333
334         fic.get();
335
336     }
337
338 }/*}}}*/
339
340 }/*}}}*/
```

FichierLu.hpp

```
/*************************************************************************
                          FichierLu  -  description
                             -------------------
    debut                : 18 nov. 2011
    copyright            : (C) 2011 par csaysset
*************************************************************************/

#ifndef FICHIERLU_HPP_
#define FICHIERLU_HPP_

#include    <fstream>
#include    <string>


namespace Reference_croisee {

using namespace Reference_croisee;

/* ================================================================================
 *        Class:  FichierLu
 *  Description:  Permet de lire un fichier en concervant son nom et le nombre de ligne
 *                deja parcourue
 * ================================================================================*/
class FichierLu : private std::ifstream {

    public:
        // ===  FUNCTION  ======================================================
        //         Name:  FichierLu
        //  Description:  Constructeur de la classe, prend en argument un chemin vers un fi
        // ================================================================================
        FichierLu ( const std::string& nomFichier = "" );




//-------------------------------------------------------------------------
//  METHODES MASQUEES
//-------------------------------------------------------------------------

        // ===  FUNCTION  ======================================================
        //         Name:  Close
        //  Description:  Ferme le fichier
        // ================================================================================
```

```
45          void close();
46
47          // ===  FUNCTION  ======================================================================
48          //         Name:  open
49          //  Description:  Ouvre un fichier
50          // =====================================================================================
51          void open( const char* filename,  ios_base::openmode mode = ios_base::in );
52
53
54          // ===  FUNCTION  ======================================================================
55          //         Name:  get
56          //  Description:  Permet de recuperer un caractere
57          // =====================================================================================
58          int get();
59
60          int peek() { return std::ifstream::peek(); }
61          bool eof() { return std::ifstream::eof(); }
62
63
64
65
66  //-------------------------------------------------------------------------
67  //  METHODES PUBLIQUES
68  //-------------------------------------------------------------------------
69
70          // ===  FUNCTION  ======================================================================
71          //         Name:  getNbLignesLues
72          //  Description:  Retourne le nombre de lignes deja lues dans le fichier
73          // =====================================================================================
74          int getNbLignesLues() const;
75
76          // ===  FUNCTION  ======================================================================
77          //         Name:  getNbLignesLues
78          //  Description:  Retourne le nom du fichier passe lors de la construction de l'obj
79          // =====================================================================================
80          std::string getNomFichier() const;
81
82
83      protected:
84          int _nbLignesLues;                      // contient le nombre de lignes lues
85          std::string _nomFichier;                // contient le nom du fichier
86  };
87
88  }/*}}}*/
89
90  #endif
```

FichierLu.cpp

```
/*************************************************************************
                           FichierLu  -  description
                             -------------------
    debut                : 18 nov. 2011
    copyright            : (C) 2011 par csaysset
*************************************************************************/

#include <fstream>
#include "FichierLu.hpp"


namespace Reference_croisee {

using namespace std;
using namespace Reference_croisee;




//-------------------------------------------------------------------------
//  CONSTRUCTEURS
//-------------------------------------------------------------------------
FichierLu::FichierLu ( const string& nomFichier ):
    _nbLignesLues( 1 )
{/*{{{*/

    if( !nomFichier.empty() ) {
        open( nomFichier.c_str() );
    }
}/*}}}*/




//-------------------------------------------------------------------------
//  METHODES MASQUEES
//-------------------------------------------------------------------------
void FichierLu::close()
{/*{{{*/

    ifstream::close();
    ifstream::clear();

    _nbLignesLues = 1;
```

```
45      _nomFichier.clear();
46  }/*}}}*/
47
48  void FichierLu::open( const char* filename,  ios_base::openmode mode )
49  {/*{{{*/
50
51      exceptions( ifstream::failbit );
52      ifstream::open( filename, mode );
53      exceptions( ifstream::badbit );
54      _nomFichier = filename;
55  }/*}}}*/
56
57  int FichierLu::get()
58  {/*{{{*/
59
60      int charactere = ifstream::get();
61
62      if ( charactere == '\n' ) {
63          _nbLignesLues++;
64      }
65
66      return charactere;
67  }/*}}}*/
68
69
70
71  //-------------------------------------------------------------------------
72  // METHODES PUBLIQUES
73  //-------------------------------------------------------------------------
74  int FichierLu::getNbLignesLues() const
75  {/*{{{*/
76
77      return _nbLignesLues;
78  }/*}}}*/
79
80  string FichierLu::getNomFichier() const
81  {/*{{{*/
82
83      return _nomFichier;
84  }/*}}}*/
85
86  }/*}}}*/
```

3.3 Module Main

Referenceur.cpp

```cpp
//============================================================================
// 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
            cout << e.what() << endl;
        }
    }
    //------------------------------------------------------------------------
    //  On charge les fichiers a referencer
    //------------------------------------------------------------------------
    vector<string> ficsReferencer;

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

    } else {
```

```
45          cerr << "Aucun fichier a referencer !" << endl;
46          return 1;
47      }
48
49      //-------------------------------------------------------------------------
50      //  On charge les mots clefs si ils sont fournis
51      //-------------------------------------------------------------------------
52      string fichierMotClef;
53
54      if( args.count( "keyword" ) ) {
55          fichierMotClef = args.get<string>( "keyword" );
56      }
57
58      //-------------------------------------------------------------------------
59      //  L'etat dans lequel mettre le programme
60      //-------------------------------------------------------------------------
61      bool mode( args.count( "exclude" ) );
62
63
64      References refs;
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 referance croisee : " << endl;
74          cerr << e.what() << endl;
75      }
76
77
78      //-------------------------------------------------------------------------
79      //  On affiche les resultats
80      //-------------------------------------------------------------------------
81      cout << refs;
82
83      return 0;
84  }/*}}}*/
```