Coralie Saysset - Romain Gérard B3301

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

Pour lancer l'execution des tests, il faut se rendre dans le répertoire des sources et taper dans un terminal la commande "make test"

2.1 Test n° 1

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

file1.cpp

```
1  // affiche le message "Hello world"
2  int main() {
3      cout <<"Hello world"<<endl;
4      cout << endl;
5      return 0;
6  }
```

file1.h

```
1  int main();
```

key1.txt

```
1  int
2  world
3  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

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

```
1   //============================================================================
2   // Name        : Ref_croisee.cpp
3   // Author      :
4   // Version     :
5   // Copyright   : Your copyright notice
6   // Description : Hello World in C++, Ansi-style
7   //============================================================================
8
9   #include <iostream>
10  #include <vector>
11
12  #include "CmdLine/cmdLine.hpp"
13  #include "References/Referenceur.hpp"
14  #include "References/References.hpp"
15
16  using namespace std;
17  using namespace Reference_croisee;
18
19  int main( int argc, char** argv )
20  {/*{{{*/
21
22  CmdLine::Arguments args;
23  CmdLine::Parser parser( "Permet de referencer des mots clefs a travers des fichiers" );
24  parser.addOption( "exclude,e",  "Inverse le fonctionnement du programme" );
25  parser.addOption( "keyword,k",  "Specifie la liste des mots clefs a utiliser", true );
26
27  try {
28      parser.parse( argc, argv, args );
29
30  } catch( exception& e ) {
31      cout << "Une erreur c'est produit durant la recuperation de la ligne de commande : "
32           << endl << e.what() << endl;
33  }
34
35  //------------------------------------------------------------------------
36  //  On charge les fichiers a referencer
37  //------------------------------------------------------------------------
38  vector<string> ficsReferencer;
39
40  if( args.count( "__args__" ) ) {
41      ficsReferencer = args.get<vector<string> >( "__args__" );
42
```

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

```
88
89          struct optValue {
90               std::string description;
91               bool hasArgument;
92               lItem* lOption;
93               sItem* sOption;
94          };
95
96          std::map< const std::string, optValue* > _lOptions;
97          std::map< const char, optValue* > _sOptions;
98          std::list<optValue> _options; // pas de vector car la zone memoire bouge
99          std::string _description;
100         std::string _freeArgs;
101
102 /*-----------------------------------------------------------------------------
103  *  Methodes privees
104  *-----------------------------------------------------------------------------*/
105
106         /* ===  FUNCTION  ======================================================================
107          *         Name:  processInput
108          *  Description:  Traite la ligne de commande
109          * =====================================================================================*/
110         void processInput( std::string::iterator& it, Arguments& args );
111
112         /* ===  FUNCTION  ======================================================================
113          *         Name:  changeState
114          *  Description:  Determine l'etat de l'objet
115          * =====================================================================================*/
116         void changeState( std::string::iterator& it );
117
118
119
120
121 };
122
123 //Surcharge de l'operateur de flux
124 std::ostream& operator<<( std::ostream& flux,  const Parser& parser );
125
126 }/*}}}*/
127 #endif
```

Parser.cpp

```cpp
// ============================================================================================
//
//       Filename:  cmdLineParser.cpp
//
//    Description:  Permet d'extraire les informations de la ligne de commande
//        Created:  06/11/2011 01:43:35
//       Compiler:  g++
//
//         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
//
// ============================================================================================


//----------------------------------------------------------------------------Include Systeme
#include    <string>
#include    <map>
#include    <utility>

//----------------------------------------------------------------------------Include Personnel
#include    "Parser.hpp"
#include    "Arguments.hpp"

namespace CmdLine {

using namespace CmdLine;
using namespace std;

/*-------------------------------------------------------------------------------------
 *  Constructeurs
 *-------------------------------------------------------------------------------------*/
Parser::Parser( string description ):
     _state( Free ), _description( description )
{}


/*-------------------------------------------------------------------------------------
 *  Methode Publiques
 *-------------------------------------------------------------------------------------*/
void Parser::AddDescription( string description )
{/*{{{*/
     _description = description;
}/*}}}*/

void Parser::AddOption( string optionName, string description, bool hasArgument )
```

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

```
91
92            flux.width( 5 );
93
94            if ( it->hasArgument )
95            {     flux << left << "arg";
96
97            }
98            else
99            {     flux << "";
100           }
101
102           flux << it->description << endl;
103
104           option.clear();
105     }
106
107 }/*}}}*/
108
109 void Parser::Parse( int argc, char** argv, Arguments& args )
110 {/*{{{*/
111
112     string cmdLine;
113
114     for ( char** it = argv + 1; it < argv + argc; it++ )
115     {           cmdLine += *it;
116           cmdLine += " ";
117     }
118
119     cmdLine += "  ";
120
121     string::iterator it = cmdLine.begin();
122
123     while ( it != cmdLine.end() )
124     {    changeState( it );
125          processInput( it, args );
126     }
127
128     if( !_freeArgs.empty() )
129     {    args._arguments.insert( make_pair( "__args__", _freeArgs ) );
130     }
131     //    for( map<string, string>::iterator it = args._arguments.begin();
132     //         it != args._arguments.end(); it++ ) {
133     //        cout << it->first << " : " << it->second << endl;
134     //    }
135
136
```

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

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

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

```
275
276         parser.Display( flux );
277
278         return flux;
279     }/*}}}*/
280
281     }/*}}}*/
```

Arguments.hpp

```
1  // NO_FORMAT ================================================================================
2  //
3  //       Filename:  cmdLineArgument.hpp
4  //
5  //    Description:  Permet de stocker les arguments passe en parametre du programme
6  //        Created:  06/11/2011 21:35:00
7  //       Compiler:  g++
8  //
9  //         Author:  Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // ==========================================================================================
12
13
14 #ifndef CmdLineArgument_HPP
15 #define CmdLineArgument_HPP
16
17 //------------------------------------------------------------------------------Include Systeme
18 #include    <string>
19 #include    <map>
20
21 //-------------------------------------------------------------------------------Include Personnel
22 #include    "Exceptions.hpp"
23 #include    "StringTo.hpp"
24
25 namespace CmdLine {
26
27 using namespace CmdLine;
28
29 /* ================================================================================================
30  *        Class:  Argument
31  *  Description:  Permet de gerer les options de la ligne de commande
32  * ================================================================================================*/
33 class Arguments {
34
35      public:
36          /* ===  FUNCTION  ======================================================================
37           *         Name:  Count
38           *  Description:  Permet de savoir si une option est presente
39           *                Il faut donner le nom long de l'option
40           * ================================================================================================*/
41          bool Count( std::string arg ) const {
42               return _arguments.count( arg );
43          }
44
```

```
45          /* ===  FUNCTION  ======================================================================
46           *         Name:  Get
47           *  Description:  Permet de recuperer une option
48           * =====================================================================================*/
49          template<typename T>
50          T Get( const char* arg ) {
51
52               if ( !_arguments.count( arg ) )
53                    {   throw InvalidKey(); }
54
55               return stringTo<T>( _arguments[arg] );
56          }
57
58          /* ===  FUNCTION  ======================================================================
59           *         Name:  Get
60           *  Description:  Permet de recuperer une option
61           * =====================================================================================*/
62          template<typename T>
63          T Get( std::string& arg ) {
64
65               if ( !_arguments.count( arg ) )
66                    {   throw InvalidKey(); }
67
68               return stringTo<T>( _arguments[arg] );
69          }
70
71
72      private:
73          /*-------------------------------------------------------------------------------
74           *  Structures de donnees
75           *-------------------------------------------------------------------------------*/
76          std::map<std::string, std::string> _arguments;
77
78          /*-------------------------------------------------------------------------------
79           *  Classe friend
80           *-------------------------------------------------------------------------------*/
81          friend class Parser;
82  };
83
84  }/*}}}*/
85
86  #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_type );
         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::string& cpAr
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

```
// ============================================================================================
//
//       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 Systeme
#include        <string>
#include        <vector>
#include        <sstream>

//-------------------------------------------------------------------------------Include Personnel
#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_type );
         return T();
    }/*}}}*/

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

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

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

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

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

```
229           std::stringstream buffer;
230           std::string cpArg = arg;
231
232           int found = cpArg.find_first_of( separator );
233           double val = 0;
234
235           while ( found != ( ( int ) std::string::npos ) )
236           {         if ( found > 0 )
237                   {         buffer << cpArg.substr( 0, found );
238                             buffer >> val;
239
240                             if ( buffer.fail() )
241                             {         throw BadConvertion();
242                           }
243
244                             conteneur.push_back( val );
245                     }
246
247                     cpArg = cpArg.substr( found + 1 );
248                     found = cpArg.find_first_of( separator );
249                     buffer.clear();
250           }
251
252           if ( arg.length() > 0 )
253           {         buffer << cpArg.substr( 0, found );
254                     buffer >> val;
255
256                     if ( buffer.fail() )
257                     {         throw BadConvertion();
258                   }
259
260                     conteneur.push_back( val );
261           }
262
263           return conteneur;
264       }/*}}}*/
265   }
266
267   #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 Systeme
#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 un motClef passe en parametre rencontre dans le fichier
        //                  nomFichier et a la ligne au conteneur
        // ====================================================================================
        void Add( const std::string& motClef, const std::string& nomFichier, int ligne );
```

```
44
45          // ===  FUNCTION  ======================================================================
46          //         Name:  Display
47          //  Description:  Permet d'afficher toutes les references dans le flux fournit en entree
48          // =====================================================================================
49          void Display( std::ostream& flux ) const;
50
51
52      protected:
53  //-------------------------------------------------------------------------
54  //  ATTRIBUTS MEMBRES
55  //-------------------------------------------------------------------------
56
57          // Structure interne du conteneur
58          std::tr1::unordered_map<std::string, std::map<std::string, std::list<int> > > _references;
59
60  };
61
62  //-------------------------------------------------------------------------
63  //  SURCHARGES OPERATEURS
64  //-------------------------------------------------------------------------
65
66  std::ostream& operator<<( std::ostream& flux,  const References& ref );
67
68  }/*}}}*/
69
70  #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 systeme
#include <iostream>

//-------------------------------------------------------------------------- include personnel
#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  }/*}}}*/
46
47  void References::Display( ostream& flux ) const
48  {/*{{{*/
49
50      tr1::unordered_map<string, map<string, list<int> > >::const_iterator itClef;
51      map<string, list<int> >::const_iterator itFic;
52      list<int>::const_iterator itLigne;
53
54
55      for ( itClef = _references.begin(); itClef != _references.end(); itClef++ )
56      {         //flux.width(15);
57          //flux << left;
58          flux << itClef->first << "\t";
59
60          for ( itFic = itClef->second.begin(); itFic != itClef->second.end(); itFic++ )
61          {         flux << itFic->first;
62
63              for ( itLigne = itFic->second.begin(); itLigne != itFic->second.end(); itLigne++ )
64              {         flux << ' ' << *itLigne;
65              }
66
67              flux << '\t';
68          }
69
70          flux << endl;
71      }
72
73  }/*}}}*/
74
75
76
77  //-------------------------------------------------------------------------
78  //  SURCHARGES OPERATEURS
79  //-------------------------------------------------------------------------
80  ostream& operator<<( ostream& flux,  const References& ref ) {
81
82      ref.Display( flux );
83
84      return flux;
85
86  }
87
88  }
```

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 Systeme
#include    <string>
#include    <vector>
#include    <tr1/unordered_set>

//------------------------------------------------------------------------------Include Personnels
#include    "FichierLu.hpp"
#include    "References.hpp"

namespace Reference_croisee {

using namespace Reference_croisee;

//-----------------------------------------------------------------------------
// Role de la classe References
// Description : Permet de visualiser la repartition de mots clefs dans une
//               collection de fichiers
//-----------------------------------------------------------------------------
class Referenceur {

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

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

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

```
137         // ===  FUNCTION  ======================================================================
138         //         Name:  lireIdentificateur
139         //  Description:  Traite les identificateurs
140         // =====================================================================================
141         void lireIdentificateur( FichierLu& fic, References& refs );
142 
143         // ===  FUNCTION  ======================================================================
144         //         Name:  lireCommentaire
145         //  Description:  Traite les commentaires
146         // =====================================================================================
147         void lireCommentaire( FichierLu& fic, References& refs );
148 
149         // ===  FUNCTION  ======================================================================
150         //         Name:  lireLiteral
151         //  Description:  Traire les chaines de caracteres
152         // =====================================================================================
153         void lireLiteral( FichierLu& fic, References& refs );
154 
155 
156 };
157 
158 }/*}}}*/
159 #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 Systeme
#include    <iostream>
#include    <limits>

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

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

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

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

```
229
230 void Referenceur::lireFlux( FichierLu& fic, References& refs )
231 {/*{{{*/
232
233     switch( _etat )
234     {
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 void Referenceur::lireCommentaire( FichierLu& fic, References& refs )
279 {/*{{{*/
280
281     fic.get();
282
283     if( fic.peek() == '/' )
284     {    while ( !fic.eof() && fic.get() != '\n' )
285          {}// bloc vide
286
287     }
288     else if( fic.peek() == '*' )
289     {    while ( !fic.eof() && ( fic.get() != '*' || fic.peek() != '/' ) )
290          {} // bloc vide
291
292         fic.get();
293     }
294
295 }/*}}}*/
296
297 void Referenceur::lireIdentificateur( FichierLu& fic, References& refs )
298 {/*{{{*/
299
300     string mot;
301     mot.append( 1, fic.get() );
302
303     while( !fic.eof() && !isSeparateur( fic.peek() ) )
304     {    mot.append( 1, fic.get() );
305     }
306
307     if( estInserable( mot ) )
308     {    refs.Add( mot, fic.GetNomFichier(), fic.GetNbLignesLues() );
309     }
310
311 }/*}}}*/
312
313 void Referenceur::lireSeparateur( FichierLu& fic, References& refs )
314 {/*{{{*/
315     fic.get();
316 }/*}}}*/
317
318 void Referenceur::lireLiteral( FichierLu& fic, References& refs )
319 {/*{{{*/
320
```

```
321     char last = fic.get();
322 
323     if( last == '"' )
324     {    while( fic.peek() != '"' || last == '\\' )
325          {    last = fic.get();
326          }
327 
328          fic.get();
329 
330     }
331     else if( last == '\'' )
332     {    while( fic.peek() != '\'' || last == '\\' )
333          {    last = fic.get();
334          }
335 
336          fic.get();
337     }
338 
339 }/*}}}*/
340 
341 }/*}}}*/
```

FichierLu.hpp

```
1                            FichierLu  -  description
2                              -------------------
3       debut                : 18 nov. 2011
4       copyright            : (C) 2011 par csaysset
5   ************************************************************************ */
6
7   #ifndef FICHIERLU_HPP_
8   #define FICHIERLU_HPP_
9
10  //------------------------------------------------------------------------Include Systeme
11  #include    <fstream>
12  #include    <string>
13
14
15  namespace Reference_croisee {
16
17  using namespace Reference_croisee;
18
19  /* ================================================================================================
20   *        Class:  FichierLu
21   *  Description:  Permet de lire un fichier en concervant son nom et le nombre de ligne
22   *                deja parcourue
23   * ================================================================================================*/
24  class FichierLu : private std::ifstream {
25
26      public:
27          // ===  FUNCTION  ======================================================================
28          //         Name:  FichierLu
29          //  Description:  Constructeur de la classe, prend en argument un chemin vers un fichier
30          // ======================================================================================
31          FichierLu ( const std::string& nomFichier = "" );
32
33
34
35
36
37  //-----------------------------------------------------------------
38  //  METHODES MASQUEES
39  //-----------------------------------------------------------------
40
41          // ===  FUNCTION  ======================================================================
42          //         Name:  Close
43          //  Description:  Ferme le fichier
44          // ======================================================================================
```

```
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:  GetNomFichier
78          //  Description:  Retourne le nom du fichier passe lors de la construction de l'objet
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

```
1  //*************************************************************************
2                              FichierLu  -  description
3                                -------------------
4      debut                : 18 nov. 2011
5      copyright            : (C) 2011 par csaysset
6  *************************************************************************/
7
8  //---------------------------------------------------------------------Include Systeme
9  #include <fstream>
10
11 //---------------------------------------------------------------------Include Personnel
12 #include "FichierLu.hpp"
13
14
15 namespace Reference_croisee {
16
17 using namespace std;
18 using namespace Reference_croisee;
19
20
21
22
23 //-----------------------------------------------------------------------------
24 //  CONSTRUCTEURS
25 //-----------------------------------------------------------------------------
26 FichierLu::FichierLu ( const string& nomFichier ):
27     _nbLignesLues( 1 )
28 {/*{{{*/
29
30     if ( !nomFichier.empty() )
31     {    open( nomFichier.c_str() );
32     }
33 }/*}}}*/
34
35
36
37
38 //-----------------------------------------------------------------------------
39 //  METHODES MASQUEES
40 //-----------------------------------------------------------------------------
41 void FichierLu::close()
42 {/*{{{*/
43
44     ifstream::close();
```

```
45      ifstream::clear();
46
47      _nbLignesLues = 1;
48      _nomFichier.clear();
49  }/*}}}*/
50
51  void FichierLu::open( const char* filename,  ios_base::openmode mode )
52  {/*{{{*/
53
54      exceptions( ifstream::failbit );
55      ifstream::open( filename, mode );
56      exceptions( ifstream::badbit );
57      _nomFichier = filename;
58  }/*}}}*/
59
60  int FichierLu::get()
61  {/*{{{*/
62
63      int charactere = ifstream::get();
64
65      if ( charactere == '\n' )
66      {         _nbLignesLues++;
67      }
68
69      return charactere;
70  }/*}}}*/
71
72
73
74  //-------------------------------------------------------------------------
75  // METHODES PUBLIQUES
76  //-------------------------------------------------------------------------
77  int FichierLu::GetNbLignesLues() const
78  {/*{{{*/
79      return _nbLignesLues;
80  }/*}}}*/
81
82  string FichierLu::GetNomFichier() const
83  {/*{{{*/
84      return _nomFichier;
85  }/*}}}*/
86
87  }/*}}}*/
```

3.3 Module Main

Ref_croisee.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\n\t" + st
        parser.AddOption( "help,h",  "Affiche ce message" );
        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 << e.what() << endl;
            cout << parser << endl;
            return 1;
        }

        if( !args.Count( "__args__" ) || args.Count( "help" ) ) {

            cout << parser << endl;
            return 1;
        }
    }

```

```
44      //--------------------------------------------------------------------
45      //  On charge les fichiers a referencer
46      //--------------------------------------------------------------------
47      vector<string> ficsReferencer;
48      ficsReferencer = args.Get<vector<string> >( "__args__" );
49
50      //--------------------------------------------------------------------
51      //  On charge les mots clefs si ils sont fournis
52      //--------------------------------------------------------------------
53      string fichierMotClef;
54
55      if( args.Count( "keyword" ) ) {
56          fichierMotClef = args.Get<string>( "keyword" );
57      }
58
59      //--------------------------------------------------------------------
60      //  L'etat dans lequel mettre le programme
61      //--------------------------------------------------------------------
62      bool mode( args.Count( "exclude" ) );
63
64
65      //--------------------------------------------------------------------
66      //  On effectue la reference croisee
67      //--------------------------------------------------------------------
68      References refs;
69      try {
70          Referenceur referenceur( fichierMotClef, mode );
71          referenceur.Referencer( ficsReferencer, refs );
72
73      } catch( exception& e ) {
74          cerr << "Une erreur est survenue durant la referance croisee : " << endl;
75          cerr << e.what() << endl;
76      }
77
78
79      //--------------------------------------------------------------------
80      //  On affiche les resultats
81      //--------------------------------------------------------------------
82      cout << refs;
83
84      return 0;
85  }/*}}}*/
```