

Références croisées

Document de réalisation

Table des matières

1	Structure de données de la STL	3
1.1	Choix pour les identificateurs	3
1.2	Choix pour les références d'occurences	3
2	Réalisation des tests	4
2.1	Test n° 1	4
	Descriptif :	4
	Résultat attendu :	4
2.2	Test n° 2	5
	Descriptif :	5
	Résultat attendu :	5
2.3	Test n° 3	6
	Descriptif :	6
	Résultat attendu :	8
2.4	Test n° 4	9
	Descriptif :	9
	Résultat attendu :	9
3	Sources de l'application	10
3.1	Makefile	10
3.2	Module CmdLine	11
	Parser.hpp	11
	Parser.cpp	14
	Arguments.hpp	21
	StringTo.hpp	23
	Exceptions.hpp	29
3.3	Module References	35
	References.hpp	35
	References.cpp	37
	Referenceur.hpp	39
	Referenceur.cpp	43
	FichierLu.hpp	51
	FichierLu.cpp	53
3.4	Module Main	55
	Ref_croisee.cpp	55

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` (ToRelease) et implémente les tables de hashages pour les classes standards de la STL (comme `string`).

```
tr1 :: std :: unordered_map < std :: string, {Occurrences} >
```

1.2 Choix pour les références d'occurences

Nous avons décidé d'utiliser une **map** de **list** pour stocker les occurrences des mots clefs. La clef de la structure `map` est le nom du fichier où apparaît l'occurrence 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'exécution 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

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

file2.h

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

key2.txt

```
1 int
2 world
3 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

```
1 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 référencer 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     } catch( exception& e ) {
30         cout << "Une erreur c'est produit durant la recuperation de la ligne de commande : "
31              << endl << e.what() << endl;
32     }
33
34
35     //-----
36     // On charge les fichiers a référencer
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 référencer !" << 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

```
1  bool      file3.cpp 60
2  catch     file3.cpp 30 72
3  char      file3.cpp 19
4  cout      file3.cpp 31 81
5  else      file3.cpp 43
6  if        file3.cpp 40 53
7  int       file3.cpp 19 19
8  namespace file3.cpp 16 17
9  return    file3.cpp 45 83
10 true      file3.cpp 25
11 try       file3.cpp 27 68
12 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'exécution 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

```
1  addOption      file4.cpp 24 25
2  argc          file4.cpp 19 28
3  args          file4.cpp 22 28 40 41 53 54 60
4  Arguments     file4.cpp 22
5  argv         file4.cpp 19 28
6  cerr         file4.cpp 44 73 74
7  CmdLine      file4.cpp 22 23
8  count        file4.cpp 40 53 60
9  display      file4.cpp 81
10 e            file4.cpp 30 32 72 74
11 endl         file4.cpp 31 32 44 73 74
12 exception    file4.cpp 30 72
13 fichierMotClef file4.cpp 51 54 69
14 ficsReferencer file4.cpp 38 41 70
15 get          file4.cpp 41 54
16 main         file4.cpp 19
17 mode         file4.cpp 60 69
18 parse       file4.cpp 28
19 Parser      file4.cpp 23
20 parser      file4.cpp 23 24 25 28
21 Reference_croisee file4.cpp 17
22 referencer   file4.cpp 70
23 References   file4.cpp 63
24 Referenceur  file4.cpp 69
25 referenceur  file4.cpp 69 70
26 refs        file4.cpp 63 70 81
27 std         file4.cpp 16
28 string      file4.cpp 38 41 51 54
29 vector      file4.cpp 38 41
30 what        file4.cpp 32 74
```

3 Sources de l'application

3.1 Makefile

```
1      CXX=g++
2      CXXFLAGS= -W -Wall -O2 -pedantic -g -pipe
3
4      SRC=$(wildcard *.cpp CmdLine/*.cpp References/*.cpp)
5      OBJ=$(SRC:.cpp=.o)
6      EXEC=referencesCroisee
7
8      $(EXEC): $(OBJ)
9              $(CXX) -o $(EXEC) $(CXXFLAGS) $^
10
11     %.o: %.cpp %.hpp
12             $(CXX) $(INCLUDES) -c $< -o $@
13
14     test: $(EXEC)
15             @(cd Tests && ./launchTests.sh)
16
17     clean:
18             rm -f *.o $(OBJ) $(EXEC)
```

3.2 Module CmdLine

Parser.hpp

```

1  // NO_FORMAT =====
2  //
3  //      Filename:  commandLineParser.hpp
4  //
5  //      Description:  Classe permettant d'extraire les informations de la ligne de commande
6  //      Created:    06/11/2011 00:50:26
7  //      Compiler:   g++
8  //
9  //      Author:    Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // =====
12
13
14 #ifndef CmdLineParser_HPP
15 #define CmdLineParser_HPP
16
17 //-----Include Systeme
18 #include <iostream>
19 #include <string>
20 #include <map>
21 #include <list>
22 #include <utility>
23
24 namespace CmdLine {
25
26 using namespace CmdLine;
27 class Arguments;
28
29
30 /* =====
31 *      Class:  Parser
32 *      Description:  Permet d'extraire et de decouper les arguments de la ligne de commande
33 *      =====*/
34 class Parser {
35
36     public:
37     /*-----
38     *      Constructeur
39     *-----*/
40     Parser( std::string = "" );
41
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:  Extraite 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

```

1  // =====
2  //
3  //      Filename:  cmdLineParser.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
14 //-----Include Systeme
15 #include    <string>
16 #include    <map>
17 #include    <utility>
18
19 //-----Include Personnel
20 #include    "Parser.hpp"
21 #include    "Arguments.hpp"
22
23 namespace CmdLine {
24
25 using namespace CmdLine;
26 using namespace std;
27
28 /*-----
29  * Constructeurs
30  *-----*/
31 Parser::Parser( string description ) :
32     _state( Free ), _description( description )
33 {}
34
35
36 /*-----
37  * Methode Publiques
38  *-----*/
39 void Parser::AddDescription( string description )
40 {
41     /*{{{*/
42     _description = description;
43     /*}}}*/
44
45 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      {
59          sIt = &( *( _sOptions.insert( make_pair( sOption.at( 0 ), ( optValue* )0 ) ).first ) );
60
61      }
62
63      optValue value = { description, hasArgument, lIt, sIt };
64      _options.push_back( value );
65
66      lIt->second = &( _options.back() );
67
68      if ( sIt )
69      {
70          sIt->second = lIt->second;
71      }
72  }/**}}}*/
73
74  void Parser::Display( ostream& flux ) const
75  {/**{{{*/
76
77      string option;
78      flux << _description << endl << endl;
79
80      flux << "Liste des arguments : " << endl;
81
82      for ( list<optValue>::const_iterator it = _options.begin(); it != _options.end(); it++ )
83      {
84          flux << "\t";
85
86          if( it->sOption )
87          {
88              option += "--";
89              option += it->sOption->first;
90              option += ", ";
91          }
92
93          option += "--";
94          flux.width( 18 );
95          flux << left << option + it->lOption->first;

```

```
91         flux.width( 5 );
92
93         if ( it->hasArgument )
94         {
95             flux << left << "arg";
96
97         }
98         else
99         {
100             flux << "";
101         }
102
103         flux << it->description << endl;
104
105         option.clear();
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     {
116         cmdLine += *it;
117         cmdLine += " ";
118     }
119
120     cmdLine += " ";
121
122     string::iterator it = cmdLine.begin();
123
124     while ( it != cmdLine.end() )
125     {
126         changeState( it );
127         processInput( it, args );
128     }
129
130     if( !_freeArgs.empty() )
131     {
132         args._arguments.insert( make_pair( "__args__", _freeArgs ) );
133     }
134
135     // for( map<string, string>::iterator it = args._arguments.begin();
136     //     it != args._arguments.end(); it++ ) {
137     //     cout << it->first << " : " << it->second << endl;
138     // }
```



```

137 }/*}}}}*/
138
139
140 /*-----
141  *  Methodes Privees
142  *-----*/
143 void Parser::changeState( string::iterator& it )
144 {/*{{{*/
145
146     if ( *it == '-' )
147     {
148         _state = ( _state == ShortCondition ) ? LongCondition : ShortCondition;
149     }
150     else if ( *it == ' ' )
151     {
152         _state = Free;
153     }
154     else if ( _state == ShortCondition )
155     {
156         _state = ShortCondition;
157     }
158     else
159     {
160         _state = FreeArgument;
161     }
162
163 }/*}}}}*/
164
165 void Parser::processInput( string::iterator& it, Arguments& args )
166 {/*{{{*/
167
168     if ( _state == Free )
169     {
170         it++;
171     }
172     else if ( _state == FreeArgument )
173     {
174         while ( *it != ' ' )
175         {
176             _freeArgs += *it;
177             it++;
178         }
179
180         _freeArgs += ',';
181     }
182     else if ( _state == LongCondition )
183     {
184         string key;
185         it++;
186
187         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         else
204         {      it++;
205
206                 while( *it != ' ' )
207                 {      value += *it;
208                         it++;
209                 }
210         }
211
212         if( value.empty() )
213         {      throw ArgumentBadFormatted();
214         }
215
216         args._arguments.insert( make_pair( key, value ) );
217
218     }
219     else if ( _state == ShortCondition )
220
221     {      if ( *it == '-' )
222             {      it++;
223             }
224
225             else
226             {      string key;
227                     key += *it;
228
```

```

229
230         if ( !_sOptions.count( *it ) )
231         {
232             throw UnknownOption( key );
233         }
234
235         sItem& item = *( _sOptions.find( *it ) );
236         key = item.second->lOption->first;
237
238         it++;
239
240         if ( !item.second->hasArgument )
241         {
242             args._arguments.insert( make_pair( key, "42" ) );
243         }
244         else
245         {
246             if ( *it != ' ' )
247             {
248                 throw ArgumentBadFormatted();
249             }
250
251             ++it;
252             string value;
253
254             while( *it != ' ' )
255             {
256                 value += *it;
257                 it++;
258             }
259
260             if( value.empty() )
261             {
262                 throw ArgumentBadFormatted();
263             }
264
265             args._arguments.insert( make_pair( key, value ) );
266         }
267     }
268 }
269
270 }/*}}}}*/
271
272 /*-----
273  *  Surcharge Operateur
274  *-----*/
275 ostream& operator<<( ostream& flux, const Parser& parser )
276 {
277     /*{{{*/

```

```
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

```

1  // =====
2  //
3  //      Filename:  StringTo.hpp
4  //
5  //      Description:  Templates pour convertir les chaines de caracteres
6  //      Created:      09/11/2011 01:00:03
7  //      Compiler:     g++
8  //
9  //      Author:      Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // =====
12
13 #ifndef CmdLineStringTo_HPP
14 #define CmdLineStringTo_HPP
15
16 #include      <string>
17 #include      <vector>
18 #include      <sstream>
19 #include      "Exceptions.hpp"
20
21
22 namespace CmdLine {
23
24 using namespace CmdLine;
25
26 #define STATIC_ASSERT( x , MSG ) typedef char __STATIC_ASSERT_##MSG[( x )?1:-1]
27
28     template<typename T>
29     T stringTo( const std::string& arg )
30     {/*****/
31
32         STATIC_ASSERT( sizeof(T) != sizeof(T), Impossible_de_convertir_l_argument_vers_ce_type );
33         return T();
34     }/*****/
35
36 /*-----
37  *  Types simples
38  *-----*/
39     template<>
40     inline int stringTo<int>( const std::string& arg )
41     {/*****/
42         std::stringstream s( arg );
43         int value = 0;
44

```

```
45         s >> value;
46
47         if ( s.fail() )
48             { throw BadConversion(); }
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 BadConversion(); }
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 BadConversion(); }
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 BadConversion(); }
```



```

91         return ( value );
92     }/*}}}}*/
93
94     template<>
95     inline std::string stringTo<std::string>( const std::string& arg )
96     {/*****/
97         return arg;
98     }/*}}}}*/
99
100
101
102     /*-----
103     *  Types composes
104     *-----*/
105     template<>
106     inline std::vector<std::string> stringTo<std::vector< std::string> >( const std::string& cpArg )
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 BadConversion(); }
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 BadConversion(); }
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 BadConversion(); }
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 BadConversion(); }
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 BadConversion(); }
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 BadConversion(); }
248
249     conteneur.push_back( val );
250 }
251
252     return conteneur;
253 }/*}}}}*/
254 }
255
256 #endif
```

Exceptions.hpp

```

1  // =====
2  //
3  //      Filename:  StringTo.hpp
4  //
5  //      Description:  Templates pour convertir les chaines de caracteres
6  //      Created:    09/11/2011 01:00:03
7  //      Compiler:   g++
8  //
9  //      Author:    Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // =====
12
13 #ifndef CmdLineStringTo_HPP
14 #define CmdLineStringTo_HPP
15
16 //-----Include Systeme
17 #include      <string>
18 #include      <vector>
19 #include      <sstream>
20
21 //-----Include Personnel
22 #include      "Exceptions.hpp"
23
24
25 namespace CmdLine {
26
27 using namespace CmdLine;
28
29 #define STATIC_ASSERT( x , MSG ) typedef char __STATIC_ASSERT_##MSG[( x )?1:-1]
30
31     template<typename T>
32     T stringTo( const std::string& arg )
33     {/*****/
34         STATIC_ASSERT( sizeof(T) != sizeof(T), Impossible_de_convertir_l_argument_vers_ce_type );
35         return T();
36     }/*****/
37
38 /*-----
39 *  Types simples
40 *-----*/
41     template<>
42     inline int stringTo<int>( const std::string& arg )
43     {/*****/
44         std::stringstream s( arg );

```

```
45         int value = 0;
46
47         s >> value;
48
49         if ( s.fail() )
50         {
51             throw BadConversion();
52         }
53
54         return value;
55     }/*}}}*/
56
57     template<>
58     inline float stringTo<float>( const std::string& arg )
59     {
60         std::stringstream s( arg );
61         float value = 0;
62
63         s >> value;
64
65         if ( s.fail() )
66         {
67             throw BadConversion();
68         }
69
70         return value;
71     }/*}}}*/
72
73     template<>
74     inline double stringTo<double>( const std::string& arg )
75     {
76         std::stringstream s( arg );
77         double value = 0;
78
79         s >> value;
80
81         if ( s.fail() )
82         {
83             throw BadConversion();
84         }
85
86         return value;
87     }/*}}}*/
88
89     template<>
90     inline bool stringTo<bool>( const std::string& arg )
91     {
92         std::stringstream s( arg );
93         int value = 0;
```

```

91         s >> value;
92
93         if ( s.fail() )
94         {
95             throw BadConversion();
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& cpArg
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         {
123             if( found > 0 )
124             {
125                 conteneur.push_back( arg.substr( 0, found ) );
126
127                 arg = arg.substr( found + 1 );
128                 found = arg.find_first_of( separator );
129             }
130
131             if ( arg.length() > 0 )
132             {
133                 conteneur.push_back( arg );
134             }
135
136             return conteneur;
137         }/*}}}}*/

```

```

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         {
151             if( found > 0 )
152             {
153                 buffer << cpArg.substr( 0, found );
154                 buffer >> val;
155
156                 if ( buffer.fail() )
157                 {
158                     throw BadConversion();
159                 }
160
161                 conteneur.push_back( val );
162             }
163
164             cpArg = cpArg.substr( found + 1 );
165             found = cpArg.find_first_of( separator );
166             buffer.clear();
167         }
168
169         if ( arg.length() > 0 )
170         {
171             buffer << cpArg.substr( 0, found );
172             buffer >> val;
173
174             if ( buffer.fail() )
175             {
176                 throw BadConversion();
177             }
178
179             conteneur.push_back( val );
180         }
181
182         return conteneur;
183     }/*****/
184
185     template<>
186         inline std::vector<double> stringTo<std::vector<double> >( const std::string& arg )
187     {/*****/

```



```

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     {
194         if ( found > 0 )
195         {
196             buffer << cpArg.substr( 0, found );
197             buffer >> val;
198
199             if ( buffer.fail() )
200             {
201                 throw BadConversion();
202             }
203
204             conteneur.push_back( val );
205         }
206
207         cpArg = cpArg.substr( found + 1 );
208         found = cpArg.find_first_of( separator );
209         buffer.clear();
210     }
211
212     if ( arg.length() > 0 )
213     {
214         buffer << cpArg.substr( 0, found );
215         buffer >> val;
216
217         if ( buffer.fail() )
218         {
219             throw BadConversion();
220         }
221
222         conteneur.push_back( val );
223     }
224
225     return conteneur;
226 }/*}}*/
227
228 template<>
229 inline std::vector<float> stringTo<std::vector<float>> >( const std::string& arg )
230 { /*{{{*/
231
232     std::vector<float> conteneur;
233     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     {
237         if ( found > 0 )
238         {
239             buffer << cpArg.substr( 0, found );
240             buffer >> val;
241
242             if ( buffer.fail() )
243             {
244                 throw BadConversion();
245             }
246
247             conteneur.push_back( val );
248         }
249
250         cpArg = cpArg.substr( found + 1 );
251         found = cpArg.find_first_of( separator );
252         buffer.clear();
253     }
254
255     if ( arg.length() > 0 )
256     {
257         buffer << cpArg.substr( 0, found );
258         buffer >> val;
259
260         if ( buffer.fail() )
261         {
262             throw BadConversion();
263         }
264
265         conteneur.push_back( val );
266     }
267
268     return conteneur;
269 }/*}}}}*/
270 }
271 #endif
```

3.3 Module References

References.hpp

```

1  // =====
2  //
3  //      Filename:  References.hpp
4  //
5  //      Description:  Permet de stocker les resultats des references croisees
6  //      Created:  18/11/2011 22:29:34
7  //      Compiler:  g++
8  //
9  //      Author:  Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // =====
12
13
14 #ifndef References_HPP
15 #define References_HPP
16
17 //-----Include Systeme
18 #include <map>
19 #include <list>
20 #include <tr1/unordered_map>
21
22 namespace Reference_croisee {
23
24 using namespace Reference_croisee;
25
26 //-----
27 // Role de la classe References
28 // Description : Permet de stocker les occurences des references croisees
29 //
30 //-----
31 class References {
32
33     public:
34     //-----
35     //  METHODES PUBLIQUES
36     //-----
37
38     // ===  FUNCTION  =====
39     //      Name:  Add
40     //      Description:  Permet d'ajouter un motClef passe en parametre rencontre dans le fichier
41     //                      nomFichier et a la ligne au conteneur
42     // =====
43     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

```

1  // =====
2  //
3  //      Filename:  References.cpp
4  //
5  //      Description:
6  //          Created:  18/11/2011 22:50:44
7  //          Compiler:  g++
8  //
9  //          Author:  Romain GERARD, romain.gerard@insa-lyon.fr
10 //
11 // =====
12
13
14 //----- include systeme
15 #include <iostream>
16
17 //----- include personnel
18 #include "References.hpp"
19
20 namespace Reference_croisee {
21
22 using namespace std;
23 using namespace Reference_croisee;
24
25
26 //-----
27 //  METHODES PUBLIQUES
28 //-----
29 void References::Add( const string& motClef, const string& nomFichier, const int ligne )
30 { /*{{{*/
31
32
33     if ( !_references.count( motClef ) )
34     {
35         _references.insert( make_pair( motClef, map<string, list<int> >() ) );
36         _references[motClef].insert( make_pair( nomFichier, list<int>( 1, ligne ) ) );
37     }
38     else if ( !_references[motClef].count( nomFichier ) )
39     {
40         _references[motClef].insert( make_pair( nomFichier, list<int>( 1, ligne ) ) );
41     }
42     else
43     {
44         _references[motClef][nomFichier].push_back( ligne );
45     }
46 }

```

```

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      {
57          //flux.width(15);
58          //flux << left;
59          flux << itClef->first << "\t";
60
61          for ( itFic = itClef->second.begin(); itFic != itClef->second.end(); itFic++ )
62          {
63              flux << itFic->first;
64
65              for ( itLigne = itFic->second.begin(); itLigne != itFic->second.end(); itLigne++ )
66              {
67                  flux << ' ' << *itLigne;
68              }
69
70              flux << '\t';
71          }
72
73          flux << endl;
74      }
75
76  }/*}}}}*/
77
78  //-----
79  //  SURCHARGES OPERATEURS
80  //-----
81  ostream& operator<<( ostream& flux,  const References& ref ) {
82
83      ref.Display( flux );
84
85      return flux;
86  }
87
88  }

```

Referenceur.hpp

```

1  // =====
2  //
3  //      Filename:  References.hpp
4  //
5  //      Description:  Interface de la classe References
6  //                   Permet de visualiser la repartition de mots clefs dans une
7  //                   collection de fichiers
8  //      Created:    15/11/2011 23:30:30
9  //      Compiler:   g++
10 //
11 //      Author:     Romain GERARD, romain.gerard@insa-lyon.fr
12 //
13 // =====
14
15
16 #ifndef Referenceur_HPP
17 #define Referenceur_HPP
18
19 //-----Include Systeme
20 #include <string>
21 #include <vector>
22 #include <tr1/unordered_set>
23
24 //-----Include Personnels
25 #include "FichierLu.hpp"
26 #include "References.hpp"
27
28 namespace Reference_croisee {
29
30 using namespace Reference_croisee;
31
32 //-----
33 // Role de la classe References
34 // Description : Permet de visualiser la repartition de mots clefs dans une
35 //              collection de fichiers
36 //-----
37 class Referenceur {
38
39     public:
40 //-----
41 //  CONSTRUCTEURS
42 //-----
43     Referenceur( const std::string fichierMotClef = std::string(),
44                 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
85     Mode _mode;
86     Etat _etat;
87
88     std::tr1::unordered_set<std::string> _motsClefs; // le conteneur des mots clefs
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

```

1  // =====
2  //
3  //      Filename:  Referenceur.cpp
4  //
5  //      Description:  Implementation de la classe Referenceur
6  //                    Permet de visualiser l'aparition de mots clefs dans une collection
7  //                    de fichiers
8  //      Created:    15/11/2011 23:37:07
9  //      Compiler:   g++
10 //
11 //      Author:    Romain GERARD, romain.gerard@insa-lyon.fr
12 //
13 // =====
14
15 using namespace std;
16
17 //-----Include Systeme
18 #include <iostream>
19 #include <limits>
20
21 //-----Include Personnel
22 #include "Referenceur.hpp"
23
24 namespace Reference_croisee {
25
26 using namespace Reference_croisee;
27
28
29
30 //-----
31 //  CONSTRUCTEURS
32 //-----
33 Referenceur::Referenceur ( const string fichierMotClef, const bool modeInverse ) :
34     _mode( modeInverse ? Inverse : Normal ), _etat( Separateur )
35 { /* {{{ */
36
37 #ifdef MAP
38     cout << "Appel au constructeur de <Referenceur>" << endl;
39 #endif
40
41     ChargerMotsClefsCpp();
42
43     if ( !fichierMotClef.empty() )
44     {
45         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     {
74         _motsClefs.insert( motRecupere );
75         fichierMotClef.ignore( numeric_limits<int>::max(), '\n' );
76         // ignore le nombre de caractere "valeur max d'un entier" jusqu'a rencontrer \n
77     }
78
79     fichierMotClef.close( );
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     {
160         fichier.open( it->c_str() );
161
162         while( !fichier.eof() )
163         {
164             changerEtat( fichier );
165             lireFlux( fichier, refs );
166         }
167
168         fichier.close();
169     }
170 }/*}}}}*/
171
172 inline void Referenceur::SetModeInverse( const bool mode )
173 {/*{{{*/
174     _mode = ( mode ) ? Inverse : Normal;
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     {
188         return false;
189     }
190
191     return ( _mode == Normal ) ? _motsClefs.count( mot ) :
192         !_motsClefs.count( mot );
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     {
215         _etat = Preprocesseur;
216     }
217     else if ( c == '/' )
218     {
219         _etat = Commentaire;
220     }
221     else if ( c == '"' || c == '\\' )
222     {
223         _etat = Literal;
224     }
225     else if ( isSeparateur( c ) )
226     {
227         _etat = Separateur;
228     }
229     else
230     {
231         _etat = MotClef;
232     }
233 }/*}}*/
```

```
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     {
272         last = fic.get();
273     }
274
275     fic.get();
```



```
275
276 }/*}}}]*/
277
278 void Referenceur::lireCommentaire( FichierLu& fic, References& refs )
279 {/*{{{*/
280
281     fic.get();
282
283     if( fic.peek() == '/' )
284     {
285         while ( !fic.eof() && fic.get() != '\n' )
286             {} // bloc vide
287
288     }
289     else if( fic.peek() == '*' )
290     {
291         while ( !fic.eof() && ( fic.get() != '*' || fic.peek() != '/' ) )
292             {} // bloc vide
293
294     }
295     fic.get();
296 }
297 }/*}}}]*/
298
299 void Referenceur::lireIdentificateur( FichierLu& fic, References& refs )
300 {/*{{{*/
301
302     string mot;
303     mot.append( 1, fic.get() );
304
305     while( !fic.eof() && !isSeparateur( fic.peek() ) )
306     {
307         mot.append( 1, fic.get() );
308     }
309
310     if( estInserable( mot ) )
311     {
312         refs.Add( mot, fic.GetNomFichier(), fic.GetNbLignesLues() );
313     }
314 }/*}}}]*/
315
316 void Referenceur::lireSeparateur( FichierLu& fic, References& refs )
317 {/*{{{*/
318     fic.get();
319 }/*}}}]*/
320
321 void Referenceur::lireLiteral( FichierLu& fic, References& refs )
322 {/*{{{*/
```

```
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 conservant 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 csayssset
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 caractere = ifstream::get();
64
65     if ( caractere == '\n' )
66     {
67         _nbLignesLues++;
68     }
69
70     return caractere;
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 }/*}}}]*/

```

3.4 Module Main

Ref_croisee.cpp

```

1  // Name      : Ref_croisee.cpp
2  // Author    :
3  // Version   :
4  // Copyright : Your copyright notice
5  // Description : Hello World in C++, Ansi-style
6  //=====
7
8  #include <iostream>
9  #include <vector>
10
11 #include "CmdLine/cmdLine.hpp"
12 #include "References/Referenceur.hpp"
13 #include "References/References.hpp"
14
15 using namespace std;
16 using namespace Reference_croisee;
17
18 int main( int argc, char** argv )
19 { /*{{{*/
20
21     CmdLine::Arguments args;
22     {
23         CmdLine::Parser parser( "Permet de référencer des mots clefs a travers des fichiers\n\t" + st
24         parser.AddOption( "help,h", "Affiche ce message" );
25         parser.AddOption( "exclude,e", "Inverse le fonctionnement du programme" );
26         parser.AddOption( "keyword,k", "Specifie la liste des mots clefs a utiliser", true );
27
28         try {
29             parser.Parse( argc, argv, args );
30
31         } catch( exception& e ) {
32             cout << e.what() << endl;
33             cout << parser << endl;
34             return 1;
35         }
36
37         if( !args.Count( "__args__" ) || args.Count( "help" ) ) {
38
39             cout << parser << endl;
40             return 1;
41         }
42     }
43

```

```
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 }/*}}}}*/
```