Département
Informatique

Dossier de Réalisation

TP-AC
Références Croisées

Binôme 3213

Tristan Pourcelot
Jordan Vincent

3IF - Groupe 2

Année scolaire 2011-2012

Institut National des Sciences Appliquées de Lyon

1 Choix généraux

1.1 Utilisation de la STL

2 Listings du programme

```
1  /*************************************************************************
                           Fichier  -  description
3                          -------------------
     debut                : Nov. 2011
5    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/
7
//-- Interface de la classe <Fichier> (fichier Fichier.h) --
9 #if ! defined ( FICHIER_H )
#define FICHIER_H
11
//------------------------------------------------------ Interfaces utilisees
13 #include <vector>
#include <string>
15 //------------------------------------------------------------ Constantes

17 //------------------------------------------------------------------ Types

19 //-------------------------------------------------------------------------
// Role de la classe <Fichier>
21 //
// La classe Fichier a pour role de representer les occurences d'un fichier
23 // pour un identificateur donne.
//
25 //-------------------------------------------------------------------------

27 class Fichier
{
29 //-------------------------------------------------------------- PUBLIC

31 public:
//----------------------------------------------------- Methodes publiques
33   void DisplayFichier() const;
   // Mode d'emploi : Affiche sur la sortie standard la liste des numeros
35   // de lignes associes au fichier.
   //
37
   bool operator==(Fichier const & unFichier) const;
39   // Mode d'emploi : renvoie vrai unFichier est egal ou this
   //
41   // Contrat : le test ne ce fait que sur l'attribut nomFic !
   //
43
   bool AddLigne( int numLigne );
45   // Mode d'emploi : Ajoute un numero de ligne au vecteur de numeros
   // de lignes. Incremente nbNumLignes.
47   // Renvoie faux si numero deja existant, vrai sinon.
   //
49   // Contrat : numLigne est strictement positif.
   //     les numeros sont supposes tries par ordre croissant
51   //     et numLigne est suppose etre plus grand que tous les
   //     numeros du vector
53

55 //------------------------------------------------- Constructeurs - destructeur

57   Fichier( const Fichier & unFichier );
   // Mode d'emploi (constructeur de copie) : construit une copie de
59   // unFichier
   // Contrat :
61
   Fichier ( string unNom );
63   // Mode d'emploi : constructeur
   // Initialise nomFic et met numLignes a NULL
65
     virtual ~Fichier ( );
67   // Mode d'emploi (destructeur) :
```

```
69 //------------------------------------------------------------------------ PRIVE
   protected:
71 //----------------------------------------------------- Methodes protegees

73 //----------------------------------------------------- Attributs proteges

75   string nomFic;
     // nom du fichier
77
     vector <int> *numLignes;
79   // Pointeur sur un tableau dynamique de numeros de lignes tries
     // par ordre  croissant.
81 };

83 //------------------------- Autres definitions dependantes de <Fichier>

85 #endif // FICHIER_H
```

../src/Fichier.h

```cpp
/*************************************************************************
                           Fichier  -  description
                             -------------------
    debut                : Nov. 2011
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//-- Realisation de la classe <Fichier> (fichier Fichier.cpp) ----

//---------------------------------------------------------------- INCLUDE

//-------------------------------------------------------- Include systeme
#include <iostream>
#include <algorithm>
using namespace std;

//------------------------------------------------------ Include personnel
#include "Fichier.h"

//------------------------------------------------------------ Constantes

//---------------------------------------------------------------- PUBLIC

//----------------------------------------------------- Methodes publiques
void Fichier::DisplayFichier() const
// Algorithme : Parcourt de numLignes et affichage des numeros de ligne
{
  cout << "\t" << nomFic;
  for (int i=0; i<numLignes->size(); i++)
  {
    cout << " " << numLignes->at(i);
  }
}

bool Fichier::AddLigne( int numLigne )
// Algorithme : Ajoute une ligne si elle n'existe pas deja.
{
  if (numLignes == NULL)
  {
    numLignes = new vector<int>;
    numLignes->push_back(numLigne);

    return 1;
  }
  else if (! binary_search (numLignes->begin(), numLignes->end(), numLigne))

  {
    numLignes->push_back(numLigne);
    return 1;
  }
  else
  {
    return 0;
  }
} // ------ Fin de AddLigne

bool Fichier::operator==(Fichier const & unFichier) const
// Algorithme : Trivial
{
  if (nomFic == unFichier.nomFic) return true;
  else return false;
} // ------ Fin de operator==

//-------------------------------------------- Constructeurs - destructeur
Fichier::Fichier ( string unNom )
// Algorithme : Trivial
{
#if defined ( MAP )
  cout << "Appel du constructeur de Fichier" << endl;
```

```
#endif
  nomFic =     unNom;
  numLignes =    NULL;
} // ------ Fin de Fichier

Fichier::Fichier ( const Fichier & unFichier )
// Algorithme : Creation d'un nouveau vector et recopie des elements
{
#if defined ( MAP )
  cout << "Appel du constructeur de copie de Fichier" << endl;
#endif
  nomFic =     unFichier.nomFic;
  numLignes =    new vector<int>;
  *numLignes =   *unFichier.numLignes;
} // ------ Fin de Fichier (Constructeur de copie)

Fichier::~Fichier ( )
// Algorithme : Trivial
{
#if defined ( MAP )
  cout << "Appel du destructeur de Fichier" << endl;
#endif
  delete numLignes;
} // ------ Fin de ~Fichier

//------------------------------------------------------------------------ PRIVE

//----------------------------------------------------------- Methodes protegees
```

../src/Fichier.cpp

```
/*************************************************************************
                          Flot  -  description
                             -------------------
    debut                : ...
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//-- Interface de la classe <Flot> (fichier Flot.h) --
#if ! defined ( FLOT_H )
#define FLOT_H

//------------------------------------------------------ Interfaces utilisees
#include <string>
#include "RefCroisees.h"

//------------------------------------------------------------------------
// Role de la classe <Flot>
//
// La classe Flot a pour role de lire et chercher les identificateurs dans
// les fichiers source.
//------------------------------------------------------------------------

class Flot
{
//-------------------------------------------------------------- PUBLIC

public:
//----------------------------------------------------- Methodes publiques

  void RemplirMotsCles(string nomFicMC, RefCroisees &uneRefMotsCles);
  // Mode d'emploi : rempli les references de mots cles avec le fichier
  // entre en parametre.
  //

  void CreerRefCrois(string nomFic, RefCroisees &uneRefMotsCles,
              RefCroisees &desRefCroisees, bool exclure = 0);
  // Mode d'emploi : lit le fichier dont le chemin est passe en parametre
  // et renseigne desRefCroisees avec les occurrences des identificateurs
  // rencontres.
  // Lorsque exclure vaut 0, les identificateurs entres dans desRefCroisees
  // seront ceux de RefMotsCles, sinon, les identificateurs seront tous les
  // identificateurs exceptes ceux de RefMotsCles.

//------------------------------------------------- Constructeurs - destructeur

//------------------------------------------------------------------ PRIVE
protected:
//----------------------------------------------------- Methodes protegees

  string FindNextId(string &phrase,bool & comActif);
  // Mode d'emploi : renvoie le premier mot de la phrase et le supprime
  // de la phrase.
  // Il supprime egalement les commentaires et les chaines de caracteres
  // Si un commentaire du type "/* */" est rencontre comActif passe a vrai
  // Si comActif est deja a vrai et que la fin de commentaire est rencontree
  // comActif devient faux

  bool IdValide( string nomId );
  // Mode d'emploi : renvoie vrai si nomId est un identificateur valide
  //
  // Contrat : nomId est compose de caracteres alphanumeriques et de '_'

};

#endif // FLOT_H
```

../src/Flot.h

```cpp
/*************************************************************************
                          Flot  -  description
                             -------------------
    debut                : Nov. 2011
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//-- Realisation de la classe <Flot> (fichier Flot.cpp) ----

//---------------------------------------------------------------- INCLUDE

//-------------------------------------------------------- Include systeme
using namespace std;
#include <iostream>
#include <fstream>
#include <locale>
//------------------------------------------------------ Include personnel
#include "Flot.h"

//------------------------------------------------------------ Constantes

//---------------------------------------------------------------- PUBLIC

//----------------------------------------------------- Methodes publiques

void Flot::RemplirMotsCles(string nomFicMC, RefCroisees &uneRefMotsCles)
// Algotithme : parcours de nomFicMC et remplissage de RefMotsCles
{
  string id,ligne;
  bool comIdPrec = false, comIdActu = false;

  ifstream fichier(nomFicMC.c_str(), ios::in);

  if (fichier)
  {
    while (getline( fichier, ligne, '\n'))
    {
      while (!ligne.empty())
      {
      id = FindNextId(ligne, comIdActu);
      if (IdValide(id) && ! comIdPrec )
        {
          uneRefMotsCles.AddReference(id);
        }
      comIdPrec = comIdActu;
      }
    }
    fichier.close();
  }
  else cerr << "/// Impossible d'ouvrir le fichier " << nomFicMC << " ///" << endl;
} // ------ Fin de RemplirMotsCles

void Flot::CreerRefCrois(string nomFic, RefCroisees &uneRefMotsCles,
                RefCroisees &desRefCroisees, bool exclure)
// Algorithme : parcours de nomFic et comparaison des identificateurs avec
// ceux de RefMotsCles.
{
  string id,ligne;
  int numLigne = 0;
  bool comIdPrec = false, comIdActu = false;

  ifstream fichier(nomFic.c_str(), ios::in);

  if (fichier)
  {
    while (getline( fichier, ligne, '\n'))
    {
      numLigne++;
      while (!ligne.empty())
```

```
      {
71      id = FindNextId(ligne, comIdActu);
        if (IdValide(id) && ! comIdPrec )
73        {
            // Les identificateurs sont ceux de desRefCroisees
75          if (! exclure && uneRefMotsCles.FindReference(id))
            {
77            desRefCroisees.AddReference(id,numLigne,nomFic.c_str());
            }
79          // Les identificateurs sont tous sauf ceux de desRefCroisees
            else if (exclure && ! uneRefMotsCles.FindReference(id))
81          {
              desRefCroisees.AddReference(id,numLigne,nomFic.c_str());
83          }
          }
85      comIdPrec = comIdActu;
        }
87    }

89    fichier.close();
    }
91  else cerr << "/// Impossible d'ouvrir le fichier " << nomFic << " ///" << endl;

93 } // ------ Fin de CreerRefCrois

95 string Flot::FindNextId(string &phrase, bool & comActif)
  // Algorithme : parcours de phrase jusqu'a trouver un caractere special
97 {
    string  mot;
99  int    i = 0;
    char  lettre = phrase[i];
101 bool  comment = comActif;

103 while ( (i < phrase.length()-1) && ( std::isalnum(lettre,std::locale()) || lettre == '
        _'))
    {
105   i++;
      lettre = phrase[i];
107 }

109 // Fin de phrase ne se terminant pas par un separateur
    if ( std::isalnum(lettre,std::locale()) || lettre == '_') i++;
111
    mot = phrase.substr(0,i);
113
    // Cas fin de commentaire '*/'
115 if (lettre == '*' && comment == true)
    {
117   if (phrase.length() >= i+2)
      {
119     if ( phrase[i+1] == '/')
        {
121       phrase.erase(0,i+2);
          comment = false;
123     }
        else phrase.erase(0,i+1);
125   }
      else phrase.erase();
127 }

129 // Cas commentaire '/*' et '*/' non trouvee
    else if (comment == true)
131 {
      phrase.erase(0,i+1);
133 }

135 // Cas des commentaires
    else if (lettre == '/' )
137 {
      if (phrase.length() >= i+2)
```

```
139     {
        // Cas des commentaires '//'
141     if ( phrase[i+1] == '/') phrase.erase();
        // Cas des commentaires '/* */'
143     else if ( phrase[i+1] == '*')
        {
145       phrase.erase(0,i+2);
          comment = true;
147     }
        else phrase.erase(0,i+1);
149   }
      else phrase.erase();
151 }

153 // Cas des chaines de caracteres
    else if (lettre == '"' && i < phrase.length()-1)
155 {
      i++;
157   lettre = phrase[i];
      while ( (i < phrase.length()-1) && lettre != '"')
159   {
        i++;
161     lettre = phrase[i];
      }
163   phrase.erase(0,i+1);
    }
165 // Debut de phrase commencant par un separateur
    else if (i == 0) phrase.erase(0,1);
167
    // cas general
169 else phrase.erase(0,i);


171
    comActif =  comment;
173 return     mot;
} //------ Fin de IdValide
175
bool Flot::IdValide( string nomId )
177 // Algorithme : Trivial
{
179   if (nomId.empty())                  return 0;
    else if (std::isdigit(nomId[0],std::locale()))  return 0;
181 else                          return 1;
} //------ Fin de IdValide
183
//------------------------------------------------ Constructeurs - destructeur
185
//------------------------------------------------------------------------ PRIVE
187
//-----------------------------------------
```

../src/Flot.cpp

```
/*************************************************************************
                          Occurrences  -  description
                             -------------------
    debut                : Nov. 2011
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//---- Interface de la classe <Occurrences> (fichier Occurrences.h) -------
#if ! defined ( OCCURRENCES_H )
#define OCCURRENCES_H

//--------------------------------------------------- Interfaces utilisees
#include "Fichier.h"

//------------------------------------------------------------------------
// Role de la classe <Occurrences>
//
// La classe Occurrences a pour role de contenir toutes les occurrences d'un
// identificateur.
//------------------------------------------------------------------------

class Occurrences
{
//------------------------------------------------------------------ PUBLIC

public:
//----------------------------------------------------- Methodes publiques
  void DisplayOccurrences () const;
  // Mode d'emploi : affiche sur la sortie standard la liste des
  // occurrences
  //

  bool AjouterOccurrence ( string nomFic, int numLigne );
  // Mode d'emploi : ajoute une occurence. Retourne faux si occurrence
  // deja existante et vrai sinon.
  //
  // Contrat : numLigne > 0

//------------------------------------------------- Constructeurs - destructeur

  Occurrences ( const Occurrences & uneOccurrences );
  // Mode d'emploi (constructeur de copie) : construit une copie de
  // uneOccurrences

  Occurrences ( );
  // Mode d'emploi :

    virtual ~Occurrences ( );
  // Mode d'emploi (destructeur) :

//------------------------------------------------------------------ PRIVE
protected:

//----------------------------------------------------- Attributs proteges
  vector <Fichier> vecOcc;
  // Tableau dynamique de Fichier, contient toutes les occurences d'un
  // identificateur sans doublons.
};

#endif // OCCURRENCES_H
```

../src/Occurrences.h

```cpp
/*************************************************************************
                          Occurrences  -  description
                             -------------------
    debut                : Nov. 2011
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//-- Realisation de la classe <Occurrences> (fichier Occurrences.cpp) ----

//------------------------------------------------------------------ INCLUDE

//-------------------------------------------------------- Include systeme
using namespace std;
#include <algorithm>
#include <iostream>
//------------------------------------------------------ Include personnel
#include "Occurrences.h"

//------------------------------------------------------------------ Constantes

//------------------------------------------------------------------ PUBLIC

//----------------------------------------------------- Methodes publiques

void Occurrences::DisplayOccurrences () const
// Algorithme: Trivial
{
  vector<Fichier>::const_iterator it;

  for ( it = vecOcc.begin(); it != vecOcc.end(); ++it)
  {
    it->DisplayFichier();
  }
}

bool Occurrences::AjouterOccurrence ( string nomFic, int numLigne )
// Algorithme :
// On recherche si il existe deja une occurrence dans un fichier,
// Puis on appelle AddLigne sur un nouveau fichier, ou sur le fichier existant
{
  vector<Fichier>::iterator it;
  Fichier fichierRecherche(nomFic);

  it = find (vecOcc.begin(), vecOcc.end(), fichierRecherche);

  // Si rien trouve
  if ( it == vecOcc.end() )
  {
    Fichier fichier( nomFic );
    fichier.AddLigne (numLigne);
    vecOcc.push_back ( fichier );
    return true;
  }
  // si trouve
  else
  {
    return it->AddLigne( numLigne );
  }

} // ------ Fin de AddOccurrence

//-------------------------------------------- Constructeurs - destructeur

Occurrences::Occurrences ( )
// Algorithme : Trivial
{
#if defined ( MAP )
  cout << "Appel du constructeur de Occurrences" << endl;
#endif
```

```
70
   } // ------ Fin de Occurrences
72
   Occurrences::Occurrences ( const Occurrences & uneOccurrences )
74 // Algorithme : Trivial
   {
76 #if defined ( MAP )
     cout << "Appel du constructeur de copie de Occurrences" << endl;
78 #endif
     vecOcc = uneOccurrences.vecOcc;
80 } // ------ Fin de Occurrences (constructeur de copie)

82 Occurrences::~Occurrences ( )
   // Algorithme : Appel de destructeur de Fichier sur chaque element
84 {
   #if defined ( MAP )
86   cout << "Appel du destructeur de Occurrences" << endl;
   #endif
88   vecOcc.erase (vecOcc.begin(),vecOcc.end());
   } // ------ Fin de ~Occurrences
90
   //-------------------------------------------------------------------- PRIVE
92
   //---------------------------------------------------------- Methodes protegees
```

../src/Occurrences.cpp

```
1  /*************************************************************************
                          RefCroisees  -  description
3                          -------------------
     debut                : Nov. 2011
5    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/
7
//---- Interface de la classe <RefCroisees> (fichier RefCroisees.h) -------
9 #if ! defined ( REFCROISEES_H )
  #define REFCROISEES_H
11
//--------------------------------------------------- Interfaces utilisees
13 #include <map>
  #include <string>
15 #include "Occurrences.h"

17 //------------------------------------------------------------ Constantes

19 //------------------------------------------------------------------ Types

21 //-------------------------------------------------------------------------
  // Role de la classe <RefCroisees>
23 //
  // La classe RefCroisees a pour role de gerer l'ensembles des references
25 // croisees.
  //
27 //-------------------------------------------------------------------------

29 class RefCroisees
  {
31 //-------------------------------------------------------------- PUBLIC

33 public:
  //----------------------------------------------------- Methodes publiques
35
    // Types personalises
37  typedef map <string, Occurrences *> TypeDicoId;
    typedef pair<string,Occurrences *>  TypePairId;
39
    void DisplayReference() const;
41  // Mode d'emploi : Affiche sur la sortie standard l'integralite de
    // dicoId et des occurrences respectives des Identificateurs.
43
    bool FindReference(const string id);
45  // Mode d'emploi : cherche l'identificaeur id dans le dictionnaire.
    // Renvoie false s'il n'existe pas, true sinon.
47
    void AddReference(string id);
49  // Mode d'emploi : Ajoute un identificateur au dictionnaire. S'il est
    // inconnu, il est cree.
51  //

53  void AddReference(string id, int numLigne, string nomFic );
    // Mode d'emploi : ajoute une reference au dictionnaire.
55  // Si l'Identificateur est inconnu, il est cree.
    // L'occurence est ensuite ajoutee.
57  //
    // Contrat : numLigne > 0
59  //

61 //------------------------------------------------- Constructeurs - destructeur
    RefCroisees ( const RefCroisees & uneRefCroisees );
63  // Mode d'emploi (constructeur de copie) : construit une copie de
    // uneRefCroisees
65  //
    // Contrat :
67  //

69  RefCroisees ();
```

```
    // Mode d'emploi : Constructeur par defaut
71  //
    // Contrat :
73  //

75    virtual ~RefCroisees ( );
    // Mode d'emploi (destructeur) :
77  //
    // Contrat :
79  //
//------------------------------------------------------------------ PRIVE
81 protected:
//------------------------------------------------------ Methodes protegees
83
//------------------------------------------------------ Attributs proteges
85
    TypeDicoId dicoId;
87  // Dictionnaire contenant des pointeurs sur les occurrences
    // correspondant a une cle de type string representant les
89  // identificateurs
};
91
//-------------------------- Autres definitions dependantes de <RefCroisees>
93
#endif // REFCROISEES_H
```

../src/RefCroisees.h

```cpp
/*************************************************************************
                           RefCroisees  -  description
                             -------------------
    debut                : Nov. 2011
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//-- Realisation de la classe <RefCroisees> (fichier RefCroisees.cpp) ----

//---------------------------------------------------------------- INCLUDE

//-------------------------------------------------------- Include systeme
using namespace std;
#include <algorithm>
#include <iostream>


//------------------------------------------------------ Include personnel
#include "RefCroisees.h"

//------------------------------------------------------------ Constantes

//---------------------------------------------------------------- PUBLIC

//----------------------------------------------------- Methodes publiques
void RefCroisees::DisplayReference() const
// Algorithme : Parcours integral du dictionnaire
{
  TypeDicoId::const_iterator it;

  for ( it = dicoId.begin(); it != dicoId.end(); ++it)
  {
    cout <<  it->first ;
    it->second->DisplayOccurrences();
    cout << endl;
  }
}

bool RefCroisees::FindReference(const string id)
// Algorithme : On parcourt l'arbre pour trouver si l'identifiant existe.
{
  TypeDicoId::iterator it;

  it = dicoId.find (id);

  if ( it == dicoId.end() )
  {
    return false;
  }
  else
  {
    return true;
  }

  return true;
} // ------ Fin de FindReference

void RefCroisees::AddReference(string id)
// Algorithme : Trivial
{
  Occurrences*  occ = NULL;
  TypePairId    myPair(id,occ);

  dicoId.insert(myPair);
} // ------ Fin de AddReference

void RefCroisees::AddReference(string id, int numLigne, string nomFic)
// Algo : insertion de l'identificateur puis de l'occurence.
{
```

```
70
   TypeDicoId::iterator       it;
72 pair<TypeDicoId::iterator,bool> pairInsert;

74 Occurrences*  occ = NULL;
   TypePairId     myPair(id,occ);
76
   pairInsert = dicoId.insert(myPair);
78
   // Nouvel identificateur
80 if (pairInsert.second == true )
   {
82   pairInsert.first->second = new Occurrences;
   }
84 pairInsert.first->second->AjouterOccurrence(nomFic, numLigne);

86 } // ------ Fin de AddReference

88 //------------------------------------------------ Constructeurs - destructeur
   RefCroisees::RefCroisees ( )
90 // Algorithme : Trivial
   {
92 #if defined ( MAP )
   cout << "Appel du constructeur de RefCroisees" << endl;
94 #endif
   } // ------ Fin de RefCroisees
96
   RefCroisees::RefCroisees ( const RefCroisees & uneRefCroisees )
98 // Algorithme : Trivial
   {
100 #if defined ( MAP )
   cout << "Appel du constructeur de copie de RefCroisees" << endl;
102 #endif
   TypeDicoId::const_iterator it;
104
   for (it = uneRefCroisees.dicoId.begin();
106   it != uneRefCroisees.dicoId.end();
      it++)
108 {
   Occurrences *occTemp = new Occurrences(*(it->second));
110 TypePairId myPair(it->first,occTemp);
   dicoId.insert ( myPair);
112 }
   } // ------ Fin de RefCroisees (constructeur de copie )
114
   RefCroisees::~RefCroisees ( )
116 // Algorithme : Trivial
   {
118 #if defined ( MAP )
   cout << "Appel du destructeur de RefCroisees" << endl;
120 #endif
   dicoId.erase (dicoId.begin(),dicoId.end());
122 } // ------ Fin de ~RefCroisees

124
   //------------------------------------------------------------------ PRIVE
126
   //------------------------------------------------------- Methodes protegees
```

../src/RefCroisees.cpp

```
/*************************************************************************
                           Main  -  Programme Principal
                             -------------------
    debut                : ...
    copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//---------- Interface de la tache <Main> (fichier Main.h) -------
#if ! defined ( MAIN_H )
#define MAIN_H

//------------------------------------------------------------------------
// Role de la tache <Main>
//
//Cette tache traite les arguments passes au programme afin de permettre
//son utilisation globale de la maniere suivante :
//
//1.        refCroisees [-e] [-k fichier_mots_cles] [nomfichier]+
//2.        refCroisees [-k fichier_mots_cles] [nomfichier]+/
//
//Options :
//  -k : Indique le fichier de mots cles a utiliser.
//  Si cette option est absente, les mots cles du C++ sont utilises.
//
//  -e : Cette option permet d'afficher les identificateurs qui ne sont pas
//  presents dans le fichier de mots cles.
//
//  En cas d'erreur d'appel, on affiche un message d'erreur ainsi qu'un
//  mode d'emploi permettant a l'utilisateur de corriger son erreur
//------------------------------------------------------------------------

////////////////////////////////////////////////////////////////  INCLUDE
//--------------------------------------------------- Interfaces utilisees

//------------------------------------------------------------ Constantes

//-------------------------------------------------------------------- Types

////////////////////////////////////////////////////////////////  PUBLIC
//---------------------------------------------------- Fonctions publiques

int main ( int argc,const char* argv[] );
// Mode d'emploi :
// Appel principal

void Usage( string aPhrase = "" );
// Mode d'emploi :
// Affiche le mode d'emploi du programme, ainsi qu'un message d'erreur personnalise


#endif // MAIN_H
```

../src/Main.h

```
/*************************************************************************
                          Main  - Programme Principal
                             -------------------
    debut                : Nov. 2011
copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
*************************************************************************/

//---------- Realisation de la tache <Main> (fichier Main.cpp) ---

//////////////////////////////////////////////////////////////////  INCLUDE
//-------------------------------------------------------- Include systeme
#include <iostream>
#include <string>
#include <cstring>

using namespace std;
//------------------------------------------------------ Include personnel
#include "Main.h"
#include "RefCroisees.h"
#include "Flot.h"

////////////////////////////////////////////////////////////////////  PRIVE
//------------------------------------------------------------ Constantes
#define ARG_ERROR 42
//-------------------------------------------------------------------- Types

////////////////////////////////////////////////////////////////////  PUBLIC
//---------------------------------------------------- Fonctions publiques
int main ( int argc,const char* argv[] )
// Algorithme : Traitement trivial des arguments
// En cas d'erreur, on renvoie un message d'erreur + Usage()
{
  //Index de l'argument traite
  int indexArg = 1;
  int i;

  string myKeywordFile;
  string myIDFile;
  string monArgTemporaire;
  Flot monFlot;
  RefCroisees mesRefCroisees;
  RefCroisees mesRefMotsCles;
  string maListeID;
  bool optionExclure = false;
  bool optionKeyWord = false;


  //In case of^^
    const int NB_MOTCLES = 63;
    string mots[NB_MOTCLES] = {"asm", "auto", "bool", "break", "case", "catch", "char",
                               "class", "const", "const_cast", "continue", "default",
                               "delete", "do", "double", "dynamic_cast", "else", "enum",
                               "explicit", "export", "extern", "false", "float", "for",
                               "friend", "goto", "if", "inline", "int", "long", "mutable
                                   ",
                               "namespace", "new", "operator", "private", "protected",
                               "public", "register", "reinterpret_cast", "return",
                               "short", "signed", "sizeof", "static", "static_cast",
                               "struct", "switch", "template", "this", "throw", "true",
                               "try", "typedef", "typeid", "typename", "union", "
                                   unsigned",
                               "using", "virtual", "void", "volatile", "wchar_t", "while
                                   "};

  // Fichier de mots cles du cpp
  string FicMotsCles = "motsClesCpp.txt";

  switch ( argc )
  {
```

```
67      case 1 :
          Usage ( "Liste d'arguments vide" ) ;
69        return ARG_ERROR;
        break;
71

73      case 2 : // Soit -k et pas de fichier soit pas d'options
          // Dans tout les cas, on utilise les mots cles du CPP
75        monArgTemporaire = argv [ 1 ];
          if ( monArgTemporaire == "-k" || monArgTemporaire == "-e" )
77          {
              Usage ( "Pas de fichier a traiter..." );
79            return ARG_ERROR;
            }
81        else
            { //On a notre fichier a traiter
83            myIDFile = argv[1];
              monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
85            monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
                          mesRefCroisees, optionExclure);
87          }
        break;
89

91      case 3 :
          // Cas d'erreur :
93        // -k + le fichier de mots cles (pas de fichier a traiter)
          // Dans tout les cas, on utilise les mots cles du CPP
95        monArgTemporaire = argv[1];
          if ( monArgTemporaire == "-k" )
97          {
              Usage ( "Pas de fichier a traiter" );
99            return ARG_ERROR;
            }
101
          if (monArgTemporaire == "-e" )
103         {
              optionExclure = true;
105           //On recupere les identifiants dans le fichier concerne
              myIDFile = argv[2];
107           monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
              monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
109                         mesRefCroisees, optionExclure);
            }
111       else
            {
113           //On recupere les identifiants dans les deux fichiers
              monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
115           for ( i = 1; i < argc; i++)
              {
117           myIDFile = argv[i];
              monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
119                         mesRefCroisees, optionExclure);
              }
121         }

123     break;

125     default :
          indexArg = 1;
127       monArgTemporaire = argv[indexArg];

129       if ( monArgTemporaire == "-e" )
            {
131           optionExclure = true;
              indexArg = indexArg + 1 ;
133         }

135       // En cas de modification de IndexArg suite a "-e"
          monArgTemporaire = argv[indexArg];
```

```
137         if ( monArgTemporaire == "-k" )
              {
139             // On recupere les mots cles
                optionKeyWord = true;
141             myKeywordFile = argv[indexArg + 1];
                monFlot.RemplirMotsCles(myKeywordFile, mesRefMotsCles);
143
                // On peut sauter un argument, on l'a traite
145             indexArg = indexArg + 2 ;
              }
147         else
              {
149             monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
              }
151
            //Et on recupere les identifiants de tout le reste
153         for ( i = indexArg; i < argc; i++ )
              {
155             myIDFile = argv[i];
                monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
157                               mesRefCroisees, optionExclure);
              }
159       break;

161   }

163   mesRefCroisees.DisplayReference();
      //mesRefCroisees.~RefCroisees();
165   return 0;

167 } //------ fin de Main

169
    void Usage ( string aPhrase )
171 //Algorithme : Trivial
    {
173   cerr << "Erreur : " << aPhrase << endl;
      cerr << "Usage : refCroisees -e [ -k fichier_keyword ] fichier1  ... fichier_n " <<
          endl;
175   cerr << "Usage : refCroisees [ -k fichier_keyword ] fichier1 ... fichier_n" << endl;
      cerr << "Options : " << endl;
177   cerr << " -k : permet d'indiquer le fichier de mots cles a utiliser" << endl;
      cerr << "Si -k est absent, les mots cles du C++ sont utilises par defaut " << endl;
179   cerr << " -e : permet d'exclure les mots cles " << endl;
      cerr << " License DWTFYWPL. Copyleft 2011 par T.Pourcelot & J.Vincent" << endl;
181 } //------ fin de Usage

183 //------ Fin de Main.cpp
```

../src/Main.cpp

3 Plan de test