



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  /*****
2                                     Fichier — description
3
4      debut                : Nov. 2011
5      copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
6      *****/
7
8  //— Interface de la classe <Fichier> (fichier Fichier.h) —
9  #if ! defined ( FICHIER_H )
10 #define FICHIER_H
11
12 //————— Interfaces utilisees
13 #include <vector>
14 #include <string>
15 //————— Constantes
16
17 //————— Types
18
19 //—————
20 // Role de la classe <Fichier>
21 //
22 // La classe Fichier a pour role de représenter les occurrences d'un fichier
23 // pour un identificateur donne.
24 //
25 //—————
26
27 class Fichier
28 {
29 //————— PUBLIC
30
31 public:
32 //————— Methodes publiques
33 void DisplayFichier() const;
34 // Mode d'emploi : Affiche sur la sortie standard la liste des numeros
35 // de lignes associes au fichier.
36 //
37 bool operator==(Fichier const & unFichier) const;
38 // Mode d'emploi : renvoie vrai unFichier est egal ou this
39 //
40 // Contrat : le test ne se fait que sur l'attribut nomFic !
41 //
42
43 bool AddLigne( int numLigne );
44 // Mode d'emploi : Ajoute un numero de ligne au vecteur de numeros
45 // de lignes. Incremente nbNumLignes.
46 // Renvoie faux si numero deja existant, vrai sinon.
47 //
48 // Contrat : numLigne est strictement positif.
49 // les numeros sont supposes tries par ordre croissant
50 // et numLigne est suppose etre plus grand que tous les
51 // numeros du vector
52
53
54 //————— Constructeurs — destructeur
55
56 Fichier( const Fichier & unFichier );
57 // Mode d'emploi (constructeur de copie) : construit une copie de
58 // unFichier
59 // Contrat :
60
61 Fichier ( string unNom );
62 // Mode d'emploi : constructeur
63 // Initialise nomFic et met numLignes a NULL
64
65 virtual ~Fichier ( );
66 // 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

```

1  /*****
3      Fichier — description
5      debut          : Nov. 2011
6      copyright      : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7  *****/
9  //— Realisation de la classe <Fichier> (fichier Fichier.cpp) —
11 //————— INCLUDE
13 //————— Include systeme
14 #include <iostream>
15 #include <algorithm>
16 using namespace std;
17 //————— Include personnel
18 #include "Fichier.h"
19 //————— Constantes
21 //————— PUBLIC
23 //————— Methodes publiques
25 void Fichier::DisplayFichier() const
26 // Algorithme : Parcourt de numLignes et affichage des numeros de ligne
27 {
28     cout << "\t" << nomFic;
29     for (int i=0; i<numLignes->size(); i++)
30     {
31         cout << " " << numLignes->at(i);
32     }
33 }
35 bool Fichier::AddLigne( int numLigne )
36 // Algorithme : Ajoute une ligne si elle n'existe pas deja.
37 {
38     if (numLignes == NULL)
39     {
40         numLignes = new vector<int>;
41         numLignes->push_back(numLigne);
42
43         return 1;
44     }
45     else if (! binary_search (numLignes->begin(), numLignes->end(), numLigne))
46     {
47         numLignes->push_back(numLigne);
48         return 1;
49     }
50     else
51     {
52         return 0;
53     }
54 } // ——— Fin de AddLigne
57 bool Fichier::operator==(Fichier const & unFichier) const
58 // Algorithme : Trivial
59 {
60     if (nomFic == unFichier.nomFic) return true;
61     else return false;
62 } // ——— Fin de operator==
63 //————— Constructeurs — destructeur
65 Fichier::Fichier ( string unNom )
66 // Algorithme : Trivial
67 {
68     #if defined ( MAP )
69     cout << "Appel du constructeur de Fichier" << endl;

```

```

71     #endif
72     nomFic =      unNom;
73     numLignes =   NULL;
74 } // ----- Fin de Fichier

75 Fichier::Fichier ( const Fichier & unFichier )
76 // Algorithme : Creation d'un nouveau vector et recopie des elements
77 {
78     #if defined ( MAP )
79     cout << "Appel du constructeur de copie de Fichier" << endl;
80     #endif
81     nomFic =      unFichier.nomFic;
82     numLignes =   new vector<int>;
83     *numLignes =  *unFichier.numLignes;
84 } // ----- Fin de Fichier (Constructeur de copie)

85 Fichier::~Fichier ( )
86 // Algorithme : Trivial
87 {
88     #if defined ( MAP )
89     cout << "Appel du destructeur de Fichier" << endl;
90     #endif
91     delete numLignes;
92 } // ----- Fin de ~Fichier

93 //-----
94 //-----
95 //----- PRIVE
96 //-----
97 //----- Methodes protegees

```

../src/Fichier.cpp

```

1  /*****
3      Flot — description
4
5      debut          : ...
6      copyright      : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7      *****/
8
9  //— Interface de la classe <Flot> (fichier Flot.h) —
10 #if ! defined ( FLOT_H )
11 #define FLOT_H
12
13 //————— Interfaces utilisees
14 #include <string>
15 #include "RefCroisees.h"
16
17 //—————
18 // Role de la classe <Flot>
19 // La classe Flot a pour role de lire et chercher les identificateurs dans
20 // les fichiers source.
21 //—————
22
23 class Flot
24 {
25 //————— PUBLIC
26
27 public:
28 //————— Methodes publiques
29
30     void RemplirMotsCles(string nomFicMC, RefCroisees &uneRefMotsCles);
31     // Mode d'emploi : rempli les references de mots cles avec le fichier
32     // entre en parametre.
33     //
34
35     void CreerRefCrois(string nomFic, RefCroisees &uneRefMotsCles,
36                       RefCroisees &desRefCroisees, bool exclure = 0);
37     // Mode d'emploi : lit le fichier dont le chemin est passe en parametre
38     // et renseigne desRefCroisees avec les occurrences des identificateurs
39     // rencontres.
40     // Lorsque exclure vaut 0, les identificateurs entres dans desRefCroisees
41     // seront ceux de RefMotsCles, sinon, les identificateurs seront tous les
42     // identificateurs exceptes ceux de RefMotsCles.
43
44 //————— Constructeurs — destructeur
45
46 //————— PRIVE
47 protected:
48 //————— Methodes protegees
49
50     string FindNextId(string &phrase, bool &comActif);
51     // Mode d'emploi : renvoie le premier mot de la phrase et le supprime
52     // de la phrase.
53     // Il supprime egalement les commentaires et les chaines de caracteres
54     // Si un commentaire du type "/* */" est rencontre comActif passe a vrai
55     // Si comActif est deja a vrai et que la fin de commentaire est rencontree
56     // comActif devient faux
57
58     bool IdValide( string nomId );
59     // Mode d'emploi : renvoie vrai si nomId est un identificateur valide
60     //
61     // Contrat : nomId est compose de caracteres alphanumeriques et de '_'
62
63 };
64
65 #endif // FLOT_H

```

../src/Flot.h

```

1  /*****
3      Flot — description
5      debut          : Nov. 2011
6      copyright      : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7  *****/
8
9  //— Realisation de la classe <Flot> (fichier Flot.cpp) —
10
11 //————— INCLUDE
12
13 //————— Include systeme
14 using namespace std;
15 #include <iostream>
16 #include <fstream>
17 #include <locale>
18 //————— Include personnel
19 #include "Flot.h"
20
21 //————— Constantes
22
23 //————— PUBLIC
24
25 //————— Methodes publiques
26
27 void Flot::RemplirMotsCles(string nomFicMC, RefCroisees &uneRefMotsCles)
28 // Algorithme : parcours de nomFicMC et remplissage de RefMotsCles
29 {
30     string id, ligne;
31     bool comIdPrec = false, comIdActu = false;
32
33     ifstream fichier(nomFicMC.c_str(), ios::in);
34
35     if (fichier)
36     {
37         while (getline( fichier, ligne, '\n'))
38         {
39             while (!ligne.empty())
40             {
41                 id = FindNextId(ligne, comIdActu);
42                 if (IdValide(id) && ! comIdPrec )
43                 {
44                     uneRefMotsCles.AddReference(id);
45                 }
46                 comIdPrec = comIdActu;
47             }
48             fichier.close();
49         }
50     }
51     else cerr << "/// Impossible d'ouvrir le fichier " << nomFicMC << " ///" << endl;
52 } // ——— Fin de RemplirMotsCles
53
54 void Flot::CreerRefCrois(string nomFic, RefCroisees &uneRefMotsCles,
55                          RefCroisees &desRefCroisees, bool exclure)
56 // Algorithme : parcours de nomFic et comparaison des identificateurs avec
57 // ceux de RefMotsCles.
58 {
59     string id, ligne;
60     int numLigne = 0;
61     bool comIdPrec = false, comIdActu = false;
62
63     ifstream fichier(nomFic.c_str(), ios::in);
64
65     if (fichier)
66     {
67         while (getline( fichier, ligne, '\n'))
68         {
69             numLigne++;
70             while (!ligne.empty())

```



```

71     {
72         id = FindNextId(ligne , comIdActu);
73         if (IdValide(id) && ! comIdPrec )
74         {
75             // Les identificateurs sont ceux de desRefCroisees
76             if (! exclure && uneRefMotsCles.FindReference(id))
77             {
78                 desRefCroisees.AddReference(id , numLigne , nomFic.c_str());
79             }
80             // Les identificateurs sont tous sauf ceux de desRefCroisees
81             else if ( exclure && ! uneRefMotsCles.FindReference(id))
82             {
83                 desRefCroisees.AddReference(id , numLigne , nomFic.c_str());
84             }
85         }
86         comIdPrec = comIdActu;
87     }
88 }
89 fichier.close();
90 }
91 else cerr << "/// Impossible d'ouvrir le fichier " << nomFic << " ///" << endl;
92 } // — Fin de CreerRefCrois
93
94 string Flot::FindNextId(string &phrase , bool & comActif)
95 // Algorithme : parcours de phrase jusqu'a trouver un caractere special
96 {
97     string mot;
98     int i = 0;
99     char lettre = phrase[i];
100     bool comment = comActif;
101
102     while ( (i < phrase.length()-1) && ( std::isalnum(lettre , std::locale()) || lettre == '_' ))
103     {
104         i++;
105         lettre = phrase[i];
106     }
107
108     // Fin de phrase ne se terminant pas par un separateur
109     if ( std::isalnum(lettre , std::locale()) || lettre == '_' ) i++;
110
111     mot = phrase.substr(0,i);
112
113     // Cas fin de commentaire '*/'
114     if (lettre == '*' && comment == true)
115     {
116         if (phrase.length() >= i+2)
117         {
118             if ( phrase[i+1] == '/')
119             {
120                 phrase.erase(0,i+2);
121                 comment = false;
122             }
123             else phrase.erase(0,i+1);
124         }
125         else phrase.erase();
126     }
127
128     // Cas commentaire '/' et '*' non trouvee
129     else if (comment == true)
130     {
131         phrase.erase(0,i+1);
132     }
133
134     // Cas des commentaires
135     else if (lettre == '/')
136     {
137         if (phrase.length() >= i+2)

```

```

139     {
140         // Cas des commentaires '//'
141         if ( phrase[i+1] == '/' ) phrase.erase();
142         // Cas des commentaires '/* */'
143         else if ( phrase[i+1] == '*' )
144         {
145             phrase.erase(0,i+2);
146             comment = true;
147         }
148         else phrase.erase(0,i+1);
149     }
150     else phrase.erase();
151 }
152
153 // Cas des chaines de caracteres
154 else if (lettre == '"' && i < phrase.length()-1)
155 {
156     i++;
157     lettre = phrase[i];
158     while ( (i < phrase.length()-1) && lettre != '"')
159     {
160         i++;
161         lettre = phrase[i];
162     }
163     phrase.erase(0,i+1);
164 }
165 // Debut de phrase commençant par un separateur
166 else if (i == 0) phrase.erase(0,1);
167
168 // cas general
169 else phrase.erase(0,i);
170
171 comActif = comment;
172 return mot;
173 } //----- Fin de IdValide
174
175 bool Flot::IdValide( string nomId )
176 // Algorithme : Trivial
177 {
178     if (nomId.empty()) return 0;
179     else if (std::isdigit(nomId[0], std::locale())) return 0;
180     else return 1;
181 } //----- Fin de IdValide
182
183 //----- Constructeurs - destructeur
184
185 //----- PRIVE
186
187 //-----

```

../src/Flot.cpp

```

2  /***** Occurrences - description *****/
4      debut          : Nov. 2011
6      copyright      : (C) 2011 par Tristan Pourcelot & Jordan Vincent
8  *****/
8  //----- Interface de la classe <Occurrences> (fichier Occurrences.h) -----
9  #if ! defined ( OCCURRENCES_H )
10 #define OCCURRENCES_H
12 //----- Interfaces utilisees
13 #include "Fichier.h"
14
15 //-----
16 // Role de la classe <Occurrences>
17 //
18 // La classe Occurrences a pour role de contenir toutes les occurrences d'un
19 // identificateur.
20 //-----
22 class Occurrences
23 {
24 //----- PUBLIC
26 public:
27 //----- Methodes publiques
28     void DisplayOccurrences () const;
29     // Mode d'emploi : affiche sur la sortie standard la liste des
30     // occurrences
31     //
32     bool AjouterOccurrence ( string nomFic, int numLigne );
33     // Mode d'emploi : ajoute une occurrence. Retourne faux si occurrence
34     // deja existante et vrai sinon.
35     //
36     // Contrat : numLigne > 0
37
38 //----- Constructeurs - destructeur
39
40     Occurrences ( const Occurrences & uneOccurrences );
41     // Mode d'emploi (constructeur de copie) : construit une copie de
42     // uneOccurrences
43
44     Occurrences ( );
45     // Mode d'emploi :
46
47     virtual ~Occurrences ( );
48     // Mode d'emploi (destructeur) :
49
50 //----- PRIVE
51 protected:
52
53 //----- Attributs proteges
54     vector <Fichier> vecOcc;
55     // Tableau dynamique de Fichier, contient toutes les occurrences d'un
56     // identificateur sans doublons.
57 };
58
59 #endif // OCCURRENCES_H

```

../src/Occurrences.h

```

2  /***** Occurrences — description *****/
4      debut          : Nov. 2011
   copyright         : (C) 2011 par Tristan Pourcelot & Jordan Vincent
6  *****/
8  //— Realisation de la classe <Occurrences> (fichier Occurrences.cpp) —
10 //————— INCLUDE
12 //————— Include systeme
   using namespace std;
14 #include <algorithm>
   #include <iostream>
16 //————— Include personnel
   #include "Occurrences.h"
18 //————— Constantes
20 //————— PUBLIC
22 //————— Methodes publiques
24 void Occurrences::DisplayOccurrences () const
26 // Algorithme: Trivial
   {
28     vector<Fichier >::const_iterator it;
30     for ( it = vecOcc.begin(); it != vecOcc.end(); ++it )
32     {
34         it->DisplayFichier();
36     }
38 bool Occurrences::AjouterOccurrence ( string nomFic, int numLigne )
39 // Algorithme :
40 // On recherche si il existe deja une occurrence dans un fichier ,
41 // Puis on appelle AddLigne sur un nouveau fichier , ou sur le fichier existant
42 {
43     vector<Fichier >::iterator it;
44     Fichier fichierRecherche(nomFic);
46     it = find (vecOcc.begin(), vecOcc.end(), fichierRecherche);
48     // Si rien trouve
49     if ( it == vecOcc.end() )
50     {
51         Fichier fichier( nomFic );
52         fichier.AddLigne (numLigne);
53         vecOcc.push_back ( fichier );
54         return true;
55     }
56     // si trouve
57     else
58     {
59         return it->AddLigne( numLigne );
60     }
61 } // ——— Fin de AddOccurrence
62 //————— Constructeurs — destructeur
64 Occurrences::Occurrences ( )
65 // Algorithme : Trivial
66 {
67     #if defined ( MAP )
68         cout << "Appel du constructeur de Occurrences" << endl;
69     #endif

```

```

70 } // ——— Fin de Occurrences
72 Occurrences::Occurrences ( const Occurrences & uneOccurrences )
74 // Algorithme : Trivial
75 {
76 #if defined ( MAP )
77     cout << "Appel du constructeur de copie de Occurrences" << endl;
78 #endif
79     vecOcc = uneOccurrences.vecOcc;
80 } // ——— Fin de Occurrences (constructeur de copie)

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

```

../src/Occurrences.cpp

```

1  /*****
3      RefCroisees  -  description
5      debut          : Nov. 2011
6      copyright      : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7  *****/
8  //----- Interface de la classe <RefCroisees> (fichier RefCroisees.h) -----
9  #if ! defined ( REFCROISEES_H )
10 #define REFCROISEES_H
11
12 //----- Interfaces utilisees
13 #include <map>
14 #include <string>
15 #include "Occurrences.h"
16
17 //----- Constantes
18
19 //----- Types
20
21 //-----
22 // Role de la classe <RefCroisees>
23 //
24 // La classe RefCroisees a pour role de gerer l'ensembles des references
25 // croisees.
26 //
27 //-----
28
29 class RefCroisees
30 {
31 //----- PUBLIC
32
33 public:
34 //----- Methodes publiques
35
36 // Types personalises
37 typedef map<string , Occurrences *> TypeDicoId;
38 typedef pair<string ,Occurrences *> TypePairId;
39
40 void DisplayReference() const;
41 // Mode d'emploi : Affiche sur la sortie standard l'integralite de
42 // dicoId et des occurrences respectives des Identificateurs.
43
44 bool FindReference(const string id);
45 // Mode d'emploi : cherche l'identificateur id dans le dictionnaire.
46 // Renvoie false s'il n'existe pas, true sinon.
47
48 void AddReference(string id);
49 // Mode d'emploi : Ajoute un identificateur au dictionnaire. S'il est
50 // inconnu, il est cree.
51 //
52
53 void AddReference(string id , int numLigne, string nomFic );
54 // Mode d'emploi : ajoute une reference au dictionnaire.
55 // Si l'Identificateur est inconnu, il est cree.
56 // L'occurence est ensuite ajoutee.
57 //
58 // Contrat : numLigne > 0
59 //
60
61 //----- Constructeurs - destructeur
62 RefCroisees ( const RefCroisees & uneRefCroisees );
63 // Mode d'emploi (constructeur de copie) : construit une copie de
64 // uneRefCroisees
65 //
66 // Contrat :
67 //
68
69 RefCroisees ();

```

```

71 // Mode d'emploi : Constructeur par défaut
72 //
73 // Contrat :
74 //
75 virtual ~RefCroisees ( );
76 // Mode d'emploi (destructeur) :
77 //
78 // Contrat :
79 //
80 //----- PRIVE
81 protected:
82 //----- Methodes protegees
83 //----- Attributs proteges
84 //
85 TypeDicoId dicoId;
86 // Dictionnaire contenant des pointeurs sur les occurrences
87 // correspondant a une cle de type string representant les
88 // identificateurs
89 };
90
91 //----- Autres definitions dependantes de <RefCroisees>
92
93 #endif // REFCROISEES_H

```

../src/RefCroisees.h

```

2  /*****
3                                     RefCroisees - description
4
5      debut                : Nov. 2011
6      copyright            : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7  *****/
8  //— Realisation de la classe <RefCroisees> (fichier RefCroisees.cpp) —
9
10 //————— INCLUDE
11
12 //————— Include systeme
13 using namespace std;
14 #include <algorithm>
15 #include <iostream>
16
17
18 //————— Include personnel
19 #include "RefCroisees.h"
20
21 //————— Constantes
22
23 //————— PUBLIC
24
25 //————— Methodes publiques
26 void RefCroisees::DisplayReference() const
27 // Algorithme : Parcours integral du dictionnaire
28 {
29     TypeDicoId::const_iterator it;
30
31     for ( it = dicoId.begin(); it != dicoId.end(); ++it )
32     {
33         cout << it->first ;
34         it->second->DisplayOccurrences();
35         cout << endl;
36     }
37 }
38
39 bool RefCroisees::FindReference(const string id)
40 // Algorithme : On parcourt l'arbre pour trouver si l'identifiant existe.
41 {
42     TypeDicoId::iterator it;
43
44     it = dicoId.find (id);
45
46     if ( it == dicoId.end() )
47     {
48         return false;
49     }
50     else
51     {
52         return true;
53     }
54
55     return true;
56 } // — Fin de FindReference
57
58 void RefCroisees::AddReference(string id)
59 // Algorithme : Trivial
60 {
61     Occurrences* occ = NULL;
62     TypePairId myPair(id, occ);
63
64     dicoId.insert(myPair);
65 } // — Fin de AddReference
66
67 void RefCroisees::AddReference(string id, int numLigne, string nomFic)
68 // Algo : insertion de l'identificateur puis de l'occurrence.
69 {

```



```

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

74     Occurrences*   occ = NULL;
76     TypePairId      myPair(id,occ);

78     pairInsert = dicoId.insert(myPair);

80     // Nouvel identificateur
81     if (pairInsert.second == true )
82     {
83         pairInsert.first->second = new Occurrences;
84     }
85     pairInsert.first->second->AjouterOccurrence(nomFic, numLigne);

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

87 //----- Constructeurs - destructeur
88 RefCroisees::RefCroisees ( )
89 // Algorithme : Trivial
90 {
91     #if defined ( MAP )
92         cout << "Appel du constructeur de RefCroisees" << endl;
93     #endif
94 } // ----- Fin de RefCroisees

95
96 RefCroisees::RefCroisees ( const RefCroisees & uneRefCroisees )
97 // Algorithme : Trivial
98 {
99     #if defined ( MAP )
100         cout << "Appel du constructeur de copie de RefCroisees" << endl;
101     #endif
102     TypeDicoId::const_iterator it;
103
104     for (it = uneRefCroisees.dicoId.begin();
105          it != uneRefCroisees.dicoId.end();
106          it++)
107     {
108         Occurrences *occTemp = new Occurrences(*(it->second));
109         TypePairId myPair(it->first,occTemp);
110         dicoId.insert ( myPair);
111     }
112 } // ----- Fin de RefCroisees (constructeur de copie )

113
114 RefCroisees::~RefCroisees ( )
115 // Algorithme : Trivial
116 {
117     #if defined ( MAP )
118         cout << "Appel du destructeur de RefCroisees" << endl;
119     #endif
120     dicoId.erase ( dicoId.begin(),dicoId.end());
121 } // ----- Fin de ~RefCroisees

122
123
124 //----- PRIVE
125
126 //----- Methodes protegees

```

../src/RefCroisees.cpp

```

1  /*****
3      Main — Programme Principal
5      debut          : ...
6      copyright      : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7      *****/
9  //————— Interface de la tache <Main> (fichier Main.h) —————
10 #if ! defined ( MAIN_H )
11 #define MAIN_H
12
13 //—————
14 // Role de la tache <Main>
15 //
16 // Cette tache traite les arguments passes au programme afin de permettre
17 // son utilisation globale de la maniere suivante :
18 //
19 // 1.      refCroisees [-e] [-k fichier_mots_cles] [nomfichier]+
20 // 2.      refCroisees [-k fichier_mots_cles] [nomfichier]+/
21 //
22 // Options :
23 // -k : Indique le fichier de mots cles a utiliser.
24 // Si cette option est absente, les mots cles du C++ sont utilises.
25 //
26 // -e : Cette option permet d'afficher les identificateurs qui ne sont pas
27 //      presents dans le fichier de mots cles.
28 //
29 // En cas d'erreur d'appel, on affiche un message d'erreur ainsi qu'un
30 // mode d'emploi permettant a l'utilisateur de corriger son erreur
31 //—————
32
33 //////////////////////////////////////// INCLUDE
34 //////////////////////////////////////// Interfaces utilisees
35
36 //////////////////////////////////////// Constantes
37
38 //////////////////////////////////////// Types
39
40 //////////////////////////////////////// PUBLIC
41 //////////////////////////////////////// Fonctions publiques
42
43 int main ( int argc, const char* argv[] );
44 // Mode d'emploi :
45 // Appel principal
46
47 void Usage( string aPhrase = "" );
48 // Mode d'emploi :
49 // Affiche le mode d'emploi du programme, ainsi qu'un message d'erreur personnalise
50
51
52
53 #endif // MAIN_H

```

../src/Main.h

```

1  /*****
3      Main — Programme Principal
5      debut : Nov. 2011
6      copyright : (C) 2011 par Tristan Pourcelot & Jordan Vincent
7      *****/
8
9  //———— Realisation de la tache <Main> (fichier Main.cpp) ———
10
11 //////////////////////////////////////////////////// INCLUDE
12 //———— Include systeme
13 #include <iostream>
14 #include <string>
15 #include <cstring>
16
17 using namespace std;
18 //———— Include personnel
19 #include "Main.h"
20 #include "RefCroisees.h"
21 #include "Flot.h"
22
23 //////////////////////////////////////////////////// PRIVE
24 //———— Constantes
25 #define ARG_ERROR 42
26
27 //———— Types
28
29 //////////////////////////////////////////////////// PUBLIC
30 //———— Fonctions publiques
31 int main ( int argc, const char* argv[] )
32 // Algorithme : Traitement trivial des arguments
33 // En cas d'erreur, on renvoie un message d'erreur + Usage()
34 {
35     //Index de l'argument traite
36     int indexArg = 1;
37     int i;
38
39     string myKeywordFile;
40     string myIDFile;
41     string monArgTemporaire;
42     Flot monFlot;
43     RefCroisees mesRefCroisees;
44     RefCroisees mesRefMotsCles;
45     string maListeID;
46     bool optionExclure = false;
47     bool optionKeyWord = false;
48
49     //In case of^^
50     const int NB_MOTCLES = 63;
51     string mots[NB_MOTCLES] = {"asm", "auto", "bool", "break", "case", "catch", "char",
52                                "class", "const", "const_cast", "continue", "default",
53                                "delete", "do", "double", "dynamic_cast", "else", "enum",
54                                "explicit", "export", "extern", "false", "float", "for",
55                                "friend", "goto", "if", "inline", "int", "long", "mutable",
56                                "namespace", "new", "operator", "private", "protected",
57                                "public", "register", "reinterpret_cast", "return",
58                                "short", "signed", "sizeof", "static", "static_cast",
59                                "struct", "switch", "template", "this", "throw", "true",
60                                "try", "typedef", "typeid", "typename", "union", "unsigned",
61                                "using", "virtual", "void", "volatile", "wchar_t", "while"};
62
63     // Fichier de mots cles du cpp
64     string FicMotsCles = "motsClesCpp.txt";
65
66     switch ( argc )
67     {

```

```

67     case 1 :
68         Usage ( "Liste d'arguments vide" ) ;
69         return ARG_ERROR;
70     break;
71
72
73     case 2 : // Soit -k et pas de fichier soit pas d'options
74         // Dans tout les cas, on utilise les mots cles du CPP
75         monArgTemporaire = argv [ 1 ];
76         if ( monArgTemporaire == "-k" || monArgTemporaire == "-e" )
77         {
78             Usage ( "Pas de fichier a traiter..." );
79             return ARG_ERROR;
80         }
81     else
82     { //On a notre fichier a traiter
83         myIDFile = argv[1];
84         monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
85         monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
86                               mesRefCroisees, optionExclure);
87     }
88     break;
89
90
91     case 3 :
92         // Cas d'erreur :
93         // -k + le fichier de mots cles (pas de fichier a traiter)
94         // Dans tout les cas, on utilise les mots cles du CPP
95         monArgTemporaire = argv [1];
96         if ( monArgTemporaire == "-k" )
97         {
98             Usage ( "Pas de fichier a traiter" );
99             return ARG_ERROR;
100         }
101
102         if (monArgTemporaire == "-e" )
103         {
104             optionExclure = true;
105             //On recupere les identifiants dans le fichier concerne
106             myIDFile = argv[2];
107             monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
108             monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
109                                   mesRefCroisees, optionExclure);
110         }
111     else
112     {
113         //On recupere les identifiants dans les deux fichiers
114         monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
115         for ( i = 1; i < argc; i++)
116         {
117             myIDFile = argv[i];
118             monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
119                                   mesRefCroisees, optionExclure);
120         }
121     }
122
123     break;
124
125     default :
126         indexArg = 1;
127         monArgTemporaire = argv[indexArg];
128
129         if ( monArgTemporaire == "-e" )
130         {
131             optionExclure = true;
132             indexArg = indexArg + 1 ;
133         }
134
135         // En cas de modification de IndexArg suite a "-e"
136         monArgTemporaire = argv[indexArg];

```

```

137     if ( monArgTemporaire == "-k" )
138     {
139         // On recupere les mots cles
140         optionKeyWord = true;
141         myKeywordFile = argv[indexArg + 1];
142         monFlot.RemplirMotsCles(myKeywordFile, mesRefMotsCles);
143
144         // On peut sauter un argument, on l'a traite
145         indexArg = indexArg + 2 ;
146     }
147     else
148     {
149         monFlot.RemplirMotsCles(FicMotsCles, mesRefMotsCles);
150     }
151
152     //Et on recupere les identifiants de tout le reste
153     for ( i = indexArg; i < argc; i++ )
154     {
155         myIDFile = argv[i];
156         monFlot.CreerRefCrois(myIDFile, mesRefMotsCles,
157                               mesRefCroisees, optionExclure);
158     }
159     break;
160 }
161
162 mesRefCroisees.DisplayReference();
163 //mesRefCroisees.~RefCroisees();
164 return 0;
165
166 } //----- fin de Main
167
168
169 void Usage ( string aPhrase )
170 //Algorithme : Trivial
171 {
172     cerr << "Erreur : " << aPhrase << endl;
173     cerr << "Usage : refCroisees -e [ -k fichier_keyword ] fichier1 ... fichier_n " <<
174         endl;
175     cerr << "Usage : refCroisees [ -k fichier_keyword ] fichier1 ... fichier_n " << endl;
176     cerr << "Options : " << endl;
177     cerr << " -k : permet d'indiquer le fichier de mots cles a utiliser " << endl;
178     cerr << "Si -k est absent, les mots cles du C++ sont utilises par default " << endl;
179     cerr << " -e : permet d'exclure les mots cles " << endl;
180     cerr << " License DWTFYWPL. Copyleft 2011 par T.Pourcelot & J.Vincent " << endl;
181 } //----- fin de Usage
182
183 //----- Fin de Main.cpp

```

../src/Main.cpp

3 Plan de test