|  |
| --- |
|  |
| Références croisées sur un ensemble de fichiers C++ |
| Présentation de la réalisation |
|  |
|  |
|  |

|  |
| --- |
| **Binôme B3302** |
| Robin Gicquel  Arnaud Mery de Montigny |
| **Le 9 décembre 2011** |

|  |
| --- |
|  |

1. **Choix de la structure de données dans la STL**

Nous voulions créer une structure de données de type arbre qui permettrait une recherche rapide d'élément à partir d'un autre élément : une clé. Nous nous sommes donc tournés vers la structure *map* qui permet de lier un élément clé unique au sein de la structure à un autre élément. Dans la classe References, nous avons introduit une variable de type *map* ayant comme clés des identificateurs pointant vers un objet de type AssocRefFichier. Ce dernier comporte aussi une variable de type map ayant comme clés des noms de fichiers où sont présents les identificateurs qui pointent à leur tour sur des ensembles d'entiers. Pour représenter cette information, nous avons choisi une structure de données de type *vector* car nous n’avons pas besoin de trier les numéros de lignes et qu’une fois insérées dans la collection, l’ordre des données ne change pas.

1. **Interface et réalisation de la classe**

* Interface

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

References - description

-------------------

début : 10 nov. 2011

copyright : (C) 2011 par Robin Gicquel et Arnaud Mery de Montigny

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//---------- Interface de la classe <References> (fichier References.h) ------

**#if** ! defined ( REFERENCES\_H\_ )

**#define** REFERENCES\_H\_

//--------------------------------------------------- Interfaces utilisées

**#include** <map>

**#include** <set>

**#include** <vector>

**#include** <string>

**#include** "AssocRefFichier.h"

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

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

**enum** Erreur

{

*ERREUR\_OUVERTURE*,

*ERREUR\_OUVERTURE\_MOTS\_CLES*,

*ERREUR\_LECTURE\_MOTS\_CLES*

};

//------------------------------------------------------------------------

// Rôle de la classe <References> :

// References est la classe qui permet de faire le lien entre les

// identificateurs et les noms de fichier qui les contiennent. Elle contient

// les fonctions permettant de lire les fichiers de mots-clés et les fichiers

// source.

//

//------------------------------------------------------------------------

**class** References

{

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

**public**:

//----------------------------------------------------- Méthodes publiques

**void**

**TraiterFichiers** ( **char** \* nomFichierMotsCles, set<string> &nomsFichiers );

// Mode d'emploi :

// Calcule les identificateurs souhaités. Le calcul dépend du paramètre

// exclureMotsCles qui précise si on veut connaitre

// les idenctificateurs présents dans les mots-clés ou les autres.

// nomFichierMotsCles est l'adresse du nom de fichier où sont présents les

// mots clés. En l'absence de ce paramètre, le programme prend une liste de

// mots clés par défaut. nomsFichiers est un objet de type Set qui comporte

// les noms de fichiers à analyser.

//

// Contrat :

// Aucun

string **AfficherResultat** ( );

// Mode d'emploi :

// Affiche Les identificateurs avec les fichiers et les numéros de lignes

// qui leurs sont liés.

//

// Contrat :

// Aucun

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

**References** ( **const** References & unReferences );

// Mode d'emploi (constructeur de copie) :

// Constructeur par copie de la classe.

//

// Contrat :

// Aucun

**References** ( );

// Mode d'emploi :

// Constructeur par défaut de la classe. Affecte la valeur false à

// l'attribut exclureMotsCles.

//

// Contrat :

// Aucun

**References** ( **bool** optExclureMotsCles );

// Mode d'emploi :

// Contructeur de la classe permettant de spécifier la valeur de

// exlureMotsCles.

//

// Contrat :

// Aucun

//

**virtual** **~References** ( );

// Mode d'emploi :

// Destructeur de la classe.

//

// Contrat :

// Aucun

//

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

**protected**:

//----------------------------------------------------- Méthodes protégées

vector<string> \* **lireFichierMotsCles** ( **char** \* nomFichier );

// Mode d'emploi :

// Lit le fichier nomFichier et retourne les mots-clés présents dans le

// fichier.

//

// Contrat :

// Aucun

**void** **lireFichier** ( **const** **char** \* nomFichier );

// Mode d'emploi :

// Lit un fichier quelconque et en discerne les identificateurs pour

// pouvoir les traiter

//

// Contrat :

// Aucun

**void** **traiterMot** ( string &mot, **const** **char** \* nomFichier, **int** numLigne );

// Mode d'emploi :

// Ajoute le mot ou non en fonction de la valeur de

// exclureMotsCles.

//

// Contrat :

// Aucun

**void** **ajouterReference** ( string &mot, **const** **char** \* nomFichier,

**int** numLigne );

// Mode d'emploi :

// Ajoute le mot comme identificateur si il n'a pas déjà

// été ajouté et le nom de fichier et le numéro de ligne.

//

// Contrat :

// Aucun

//----------------------------------------------------- Attributs protégés

map<string, AssocRefFichier> references;

// references possède comme clés les identificateurs retenus et pointent

// vers une instance de AssocRefFichier

vector<string> \* motsCles;

// Tableau dynamique de mots-clés

**bool** exclureMotsCles;

// Précise si on veut lister uniquement les mots-clés ou bien tous les

// identificateurs sauf les mots-clés.

};

//--------------------------- Autres définitions dépendantes de <References>

**#endif** // REFERENCES\_H\_

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AssocRefFichier - description

-------------------

début : 10 nov. 2011

copyright : (C) 2011 par Robin Gicquel et Arnaud Mery de Montigny

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//--- Interface de la classe <AssocRefFichier> (fichier AssocRefFichier.h) ---

**#if** ! defined ( ASSOCREFFICHIER\_H\_ )

**#define** ASSOCREFFICHIER\_H\_

//--------------------------------------------------- Interfaces utilisées

**#include** <vector>

**#include** <map>

**#include** <string>

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

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

//------------------------------------------------------------------------

// Rôle de la classe AssocRefFichier :

// La classe permet de représenter une association entre un nom de fichier et

// une liste de numéro de lignes (aussi appelés références).

//------------------------------------------------------------------------

**class** AssocRefFichier

{

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

**public**:

//----------------------------------------------------- Méthodes publiques

**void** **AjouterReference** ( string nomFichier, **int** numLigne );

// Mode d'emploi :

// Ajoute un numéro de ligne pour un fichier. Si le fichier n'est pas

// encore référencé, il est ajouté en tant que clé à la map.

//

// Contrat :

// Aucun

//

string **AfficherFichiers** ( );

// Mode d'emploi :

// Retourne une chaine de caractère représentant les fichiers avec

// les numéros de lignes qui leurs sont liés.

//

// Contrat :

// Aucun

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

**AssocRefFichier** ( **const** AssocRefFichier & unAssocRefFichier );

// Mode d'emploi (constructeur de copie) :

// Constructeur par copie de la classe.

//

// Contrat :

// Aucun

**AssocRefFichier** ( );

// Mode d'emploi :

// Constrcuteur par défaut de la classe

//

// Contrat :

// Aucun

**virtual** **~AssocRefFichier** ( );

// Mode d'emploi : Destructeur de la classe

//

// Contrat :

// Aucun

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

**protected**:

//----------------------------------------------------- Méthodes protégées

string **afficherLignes** ( vector<**int**> & lignes );

// Mode d'emploi :

// Retourne une chaine de caractère représentant les numéros de lignes

// contenus dans le vector passé en paramètre.

//

// Contrat :

// Aucun

//----------------------------------------------------- Attributs protégés

map<string, vector<**int**> > fichiers;

// fichiers possède comme clés des noms de fichiers et comme valeur un

// tableau dynamique d'entiers représentant des numéros de ligne

};

//------------------------ Autres définitions dépendantes de <AssocRefFichier>

**#endif** // ASSOCREFFICHIER\_H\_

* Réalisation

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

References - description

-------------------

début : 10 nov. 2011

copyright : (C) 2011 par Robin Gicquel et Arnaud Mery de Montigny

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//------- Réalisation de la classe <References> (fichier References.cpp) -----

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

//-------------------------------------------------------- Include système

**using** **namespace** std;

**#include** <fstream>

**#include** <map>

**#include** <vector>

**#include** <string>

**#include** <algorithm>

**#include** <utility>

**#include** <cstring>

**#include** <cstdlib>

//------------------------------------------------------ Include personnel

**#include** "References.h"

**#include** "AssocRefFichier.h"

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

**static** **const** **char** DELIM[] = {

' ', '\t', '\r', '\n', ';', ':', ',', '.', '<', '>', '=', '{', '}', '(',

')', '!', '-', '+', '/', '\*', '&', '|', '%', '$', '#', '[', ']' };

**static** **const** **char** TAILLE\_DELIM = 27;

**static** **const** string MOTS\_CLES\_C[] = {

"asm", "auto", "bool", "break", "case", "catch", "char", "class",

"const", "const\_char", "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" };

**static** **const** **int** TAILLE\_MOTS\_CLES\_C = 63;

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

//----------------------------------------------------- Méthodes publiques

**void** **References::TraiterFichiers** ( **char** \* nomFichierMotsCles,

set<string> &nomsFichiers )

// Algorithme : Trivial

{

**if** ( nomFichierMotsCles != NULL )

{

lireFichierMotsCles ( nomFichierMotsCles );

}

**else**

{

// remplissage du vector à partir du tableau constant de mots-clés

motsCles = **new** vector<string> ( MOTS\_CLES\_C,

MOTS\_CLES\_C + TAILLE\_MOTS\_CLES\_C );

}

**for** ( set<string>::iterator it = nomsFichiers.begin ( );

it != nomsFichiers.end ( ); it++ )

{

**const** **char** \* truc = it->c\_str ( );

lireFichier ( truc );

}

} //----- Fin de TraiterFichiers

string **References::AfficherResultat** ( )

// Algorithme : Trivial

{

string str = "";

**for** ( map<string, AssocRefFichier>::iterator it = references.begin ( );

t != references.end ( ); it++ )

{

str += it->first + it->second.AfficherFichiers ( ) + '\n';

}

**return** str;

} //----- Fin de AfficherResultat

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

**References::References** ( **const** References & unReferences )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au constructeur de copie de <References>" << endl;

**#endif**

} //----- Fin de References (constructeur de copie)

**References::References** ( )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au constructeur par défaut de <References>" << endl;

**#endif**

} //----- Fin de References (contructeur par défaut)

**References::References** ( **bool** optExclureMotsCles ) :

exclureMotsCles ( optExclureMotsCles )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au constructeur de <References>" << endl;

**#endif**

} //----- Fin de References

**References::~References** ( )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au destructeur de <References>" << endl;

**#endif**

**delete** motsCles;

} //----- Fin de ~References

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

//----------------------------------------------------- Méthodes protégées

vector<string> \* **References::lireFichierMotsCles** ( **char** \* nomFichier )

// Algorithme : Lecture ligne par ligne d'un fichier

{

string str;

**char** c;

motsCles = **new** vector<string>;

ifstream lecture;

lecture.open ( nomFichier );

**if** ( lecture.fail ( ) )

{

**throw** *ERREUR\_OUVERTURE\_MOTS\_CLES*;

}

**while** (!lecture.eof ( ))

{

str = "";

// Ajoute les caractères en provenance du flux jusqu'à trouver une

// fin de ligne

**while** ((c = lecture.get ( )) != '\n' && c != '\r' && c != -1)

{

str.push\_back ( c );

}

// Vide le "buffer" des éventuels '\n' et '\r' restants

**while** ((c = lecture.peek ( )) == '\n' || c == '\r')

{

lecture.get ( );

}

// La fonction find() renvoie la constante npos si la chaîne n'est pas

// trouvée

**if** ( str.find ( ' ' ) == string::*npos*

|| str.find ( ',' ) == string::*npos*

|| str.find ( ';' ) == string::*npos*

|| str.find ( '\t' ) == string::*npos* )

{

**throw** *ERREUR\_LECTURE\_MOTS\_CLES*;

}

motsCles->push\_back ( str );

}

lecture.close ( );

**return** motsCles;

} //----- Fin de lireFichierMotsCles

**void** **References::lireFichier** ( **const** **char** \* nomFichier )

// Algorithme :

// Lecture caractère par caratère d'un fichier avec détection des

// commentaires. Dès qu'un mot est isolé, il est recherché dans la liste des

// mots clés et ajouté aux références en fonction des options fournies.

{

ifstream lecture;

string mot = "";

**int** numLigne = 1;

**int** c;

**int** c2;

**char** carAttendu1 = -1;

**char** carAttendu2 = -1;

lecture.open ( nomFichier );

**if** ( lecture.fail ( ) )

{

**throw** *ERREUR\_OUVERTURE*;

}

**while** (!lecture.eof ( ))

{

c = lecture.get ( );

**if** ( carAttendu1 == -1 )

{

**if** ( c == '/' )

{

c2 = lecture.peek ( );

**if** ( c2 == '/' )

{

carAttendu1 = '\n';

lecture.get ( );

}

**else** **if** ( c2 == '\*' )

{

carAttendu1 = '\*';

carAttendu2 = '/';

lecture.get ( );

}

}

**else** **if** ( c == '"' )

{

carAttendu1 = '"';

}

**else** **if** ( c == '\'' )

{

carAttendu1 = '\'';

}

**else** **if** ( find ( DELIM, DELIM + TAILLE\_DELIM, c )

== DELIM + TAILLE\_DELIM ) // Recherche du mot lu dans les mots-clés

{

mot.push\_back ( c );

}

**else**

{

**if** ( mot != "" )

{

traiterMot ( mot, nomFichier, numLigne );

mot.clear ( );

}

}

}

**else**

{

**if** ( c == carAttendu1 )

{

**if** ( carAttendu2 == -1 )

{

carAttendu1 = -1;

}

**else**

{

**if** ( lecture.peek ( ) == carAttendu2 )

{

carAttendu1 = -1;

carAttendu2 = -1;

**if** ( lecture.get ( ) == '\n' )

{

numLigne++;

}

}

}

}

}

**if** ( lecture.peek ( ) == '\n' )

{

numLigne++;

}

}

} //----- Fin de lireFichier

**void** **References::traiterMot** ( string &mot, **const** **char** \* nomFichier,

**int** numLigne )

// Algorithme : Trivial, point délicat détaillé dans le corps

{

**const** **char** \* motC = mot.c\_str ( );

**char** \*\* retourStrtod = **new** **char**\*;

// Pour vérifier si motC n'est pas un nombre, on teste s'il s'agit d'un

// réel. Pour cela, on utilise la fonction C strtod qui renvoie 0.0 si la

// chaîne passée en paramètre ne correspond pas à un nombre réel. Pour

// différencier la lecture de la chaîne "0" et celle d'une chaîne quelconque

// ne correspondant pas à un nombre réel, la fonction affecte l'adresse du

// premier paramètre passé à la valeur du second paramètre passé. Ainsi, en

// vérifiant que \*retourStrtod est égal à motC, on est assuré que le mot est

// ou n'est un nombre.

**if** ( strtod ( motC, retourStrtod ) == 0.0 && \*retourStrtod == motC )

{

**if** ( find ( motsCles->begin ( ), motsCles->end ( ), mot )

!= motsCles->end ( ) )

{

**if** ( !exclureMotsCles )

{

ajouterReference ( mot, nomFichier, numLigne );

}

}

**else** **if** ( exclureMotsCles )

{

ajouterReference ( mot, nomFichier, numLigne );

}

}

**delete** retourStrtod;

} //----- Fin de traiterMot

**void** **References::ajouterReference** ( string &mot, **const** **char** \* nomFichier,

**int** numLigne )

// Algorithme : Trivial

{

AssocRefFichier \* assoc = **new** AssocRefFichier ( );

pair<map<string, AssocRefFichier>::iterator, **bool**> paire;

paire = references.insert (

pair<string, AssocRefFichier> ( mot, \*(assoc) ) );

map<string, AssocRefFichier>::iterator it = paire.first;

it->second.AjouterReference ( nomFichier, numLigne );

**delete** assoc;

} //----- Fin de ajouterReference

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AssocRefFichier - description

-------------------

début : 10 nov. 2011

copyright : (C) 2011 par Robin Gicquel et Arnaud Mery de Montigny

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//- Réalisation de la classe <AssocRefFichier> (fichier AssocRefFichier.cpp) -

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

//-------------------------------------------------------- Include système

**using** **namespace** std;

**#include** <iostream>

**#include** <sstream>

**#include** <vector>

**#include** <string>

**#include** <map>

//------------------------------------------------------ Include personnel

**#include** "AssocRefFichier.h"

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

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

//----------------------------------------------------- Méthodes publiques

**void** **AssocRefFichier::AjouterReference** ( string nomFichier, **int** numLigne )

// Algorithme : Trivial

{

vector<**int**> \* vec = **new** vector<**int**>;

pair<map<string, vector<**int**> >::iterator, **bool**> paire;

paire = fichiers.insert (

pair<string, vector<**int**> > ( nomFichier, \*(vec) ) );

map<string, vector<**int**> >::iterator it = paire.first;

it->second.push\_back(numLigne);

// le contenu de vec a été copié, on doit donc libérer la mémoire

**delete** vec;

} //----- Fin de ajouterLigne

string **AssocRefFichier::AfficherFichiers** ( )

// Algorithme : Trivial

{

string str = "";

**for** ( map<string, vector<**int**> >::iterator it = fichiers.begin ( );

it != fichiers.end ( ); it++ )

{

str += '\t' + it->first + afficherLignes (it->second);

}

**return** str;

}

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

**AssocRefFichier::AssocRefFichier** ( **const** AssocRefFichier & unAssocRefFichier )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au constructeur de copie de <AssocRefFichier>" << endl;

**#endif**

// appel au constructeur par copie de map

fichiers = unAssocRefFichier.fichiers;

} //----- Fin de AssocRefFichier (constructeur de copie)

**AssocRefFichier::AssocRefFichier** ( )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au constructeur de <AssocRefFichier>" << endl;

**#endif**

} //----- Fin de AssocRefFichier

**AssocRefFichier::~AssocRefFichier** ( )

// Algorithme : Trivial

{

**#ifdef** MAP

cout << "Appel au destructeur de <AssocRefFichier>" << endl;

**#endif**

} //----- Fin de ~AssocRefFichier

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

//----------------------------------------------------- Méthodes protégées

string **AssocRefFichier::afficherLignes** (vector<**int**> & lignes)

// Algorithme : Trivial

{

string str = "";

ostringstream oss;

**for** ( vector<**int**>::iterator it = lignes.begin ( ); it != lignes.end ( );

it++ )

{

oss << \*(it);

str += ' ' + oss.str ( );

oss.str ( "" );

}

**return** str;

} //----- Fin de AfficherLignes

* Main

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Main - description

-------------------

début : 10 nov. 2011

copyright : (C) 2011 par Robin Gicquel et Arnaud Mery de Montigny

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//---------- Réalisation du module <Main> (fichier Main.cpp) -----

///////////////////////////////////////////////////////////////// INCLUDE

//-------------------------------------------------------- Include système

**using** **namespace** std;

**#include** <iostream>

**#include** <set>

**#include** <string>

//------------------------------------------------------ Include personnel

**#include** "References.h"

/////////////////////////////////////////////////////////////////// PRIVE

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

**static** **const** **int** RET\_ERR\_ARGS = 1;

**static** **const** **int** RET\_ERR\_LECTURE = 2;

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

//---------------------------------------------------- Variables statiques

//------------------------------------------------------ Fonctions privées

////////////////////////////////////////////////////////////////// PUBLIC

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

**int** **main** ( **int** argc, **char** \*\* argv )

// Algorithme :

// Lecture des paramètres fournis à l'exécution, vérification de

// leur validité puis lancement des opérations de traitement de fichiers avec

// les options adéquates.

{

**if** ( argc < 2 )

{

cout << "Nombres d'arguments insuffisant" << **endl**;

**return** RET\_ERR\_ARGS;

}

**bool** exclureMotsCles = **false**;

**bool** fichierMotsClesAttendu = **false**;

**int** positionAttendue;

**char** \* nomFichierMotsCles = NULL;

set<string> nomsFichiers;

// Traitement des arguments fournis

**for** ( **int** i = 1; i < argc; i++ )

{

**if** ( argv[i][0] == '-' )

{

**if** ( argv[i][1] == 'e' && argv[i][2] == 0 )

{

**if** ( i == 1 )

{

**if** ( argc < 3 )

{

cerr << "Aucun nom de fichier fourni" << **endl**;

**return** RET\_ERR\_ARGS;

}

exclureMotsCles = **true**;

}

**else**

{

cerr << "Position de l'argument -e invalide" << **endl**;

**return** RET\_ERR\_ARGS;

}

}

**else** **if** ( argv[i][1] == 'k' && argv[i][2] == 0 )

{

**if** ( exclureMotsCles )

{

positionAttendue = 2;

}

**else**

{

positionAttendue = 1;

}

**if** ( i == positionAttendue && argv[i + 1][0] != '-' )

{

fichierMotsClesAttendu = **true**;

}

**else** **if** ( argv[i + 1][0] == '-' )

{

cerr << "FichierMotCles attendu apres l'option -k" << **endl**;

**return** RET\_ERR\_ARGS;

}

**else**

{

cerr << "Position de l'argument -k invalide" << **endl**;

**return** RET\_ERR\_ARGS;

}

**if** ( argc < positionAttendue + 3 )

{

cerr << "Nombre d'arguments insuffisant" << **endl**;

**return** RET\_ERR\_ARGS;

}

}

**else**

{

cerr << "Option inconnue" << **endl**;

**return** RET\_ERR\_ARGS;

}

}

**else**

{

**if** ( fichierMotsClesAttendu )

{

nomFichierMotsCles = argv[i];

fichierMotsClesAttendu = **false**;

}

**else**

{

nomsFichiers.insert ( argv[i] );

}

}

}

References ref ( exclureMotsCles );

**try**

{

ref.TraiterFichiers ( nomFichierMotsCles, nomsFichiers );

}

**catch** (Erreur &e)

{

**if** ( e == *ERREUR\_OUVERTURE* )

{

cerr << "Erreur lors de l'ouverture d'un fichier source" << **endl**;

}

**else** **if** ( e == *ERREUR\_LECTURE\_MOTS\_CLES* )

{

cerr << "Erreur de syntaxe dans le fichier de mots-clés" << **endl**;

}

**else** **if** ( e == *ERREUR\_OUVERTURE\_MOTS\_CLES* )

{

cerr << "Erreur lors de l'ouverture du fichier de mots-clés"

<< **endl**;

}

**return** RET\_ERR\_LECTURE;

}

cout << ref.AfficherResultat ( ) << **endl**;

**return** 0;

} //----- fin de main

1. **Réalisation du plan de tests fonctionnels**

* Fichiers nécessaires pour les besoins des tests avec leur contenu :

**keywords.txt:**

int

world

template

**keywords2.txt** est vide.

**keywords3.txt:**

int main();int entier;

**keywords4.txt :**

//commentaire

int a;

/\*voici

un autre commentaire\*/

char b = 'a';

string c = "Hello";

**fileTest1.txt:**

// affiche le message "Hello world"

int main() {

cout<<"Hello world"<<endl;

cout<<endl;

return 0;

}

**fileTest2.txt:**

int main();

**fileTest3.txt:**

int main();int entier;

**fileTest4.txt:**

//commentaire

int a;

/\*voici

un autre commentaire\*/

char b = 'a';

string c = "Hello";

* Tests réalisés en concordance avec le plan de test dans le document de spécification et conception :

**Test1.1**:TP\_STL.exe -k keywords.txt fileTest1.txt -e

**Test1.2:**TP\_STL.exe -k keywords.txt -e fileTest1.txt

**Test1.3**:TP\_STL.exe -k -e keywords.txt fileTest1.txt

**Test1.4**:TP\_STL.exe -e keywords.txt -k fileTest1.txt

**Test1.5**:TP\_STL.exe

**Test1.6**:TP\_STL.exe -e -k keywords.txt

**Test1.7**:TP\_STL.exe -f -k keywords.txt

**Test2**:TP\_STL.exe fileTest2.txt fileTest2.txt

**Test3**:TP\_STL.exe fileTest3.txt

**Test4**:TP\_STL.exe -k keywords2.txt fileTest1.txt

**Test5**:TP\_STL.exe fileTest4.txt

**Test6**:TP\_STL.exe -e -k keywords3.txt fileTest1.txt

**Test7**:TP\_STL.exe -e -k key.txt fileTest1.txt

**Test8**:TP\_STL.exe -e fileTest1.txt fileTest2.txt fileTest3.txt toto.txt

**Test9**:TP\_STL.exe -e -k keywords.txt fileTest1.txt fileTest2.txt

**Test10**:TP\_STL.exe -e fileTest1.txt fileTest2.txt

**Test11**:TP\_STL.exe -k keywords.txt fileTest1.txt fileTest2.txt

**Test12**:TP\_STL.exe fileTest1.txt fileTest2.txt

* Résultat des Tests

**Test1.1**: Position de l'argument -e invalide

**Test1.2:** Position de l'argument -e invalide

**Test1.3**: FichierMotCles attendu apres l'option -k

**Test1.4**: Position de l'argument -k invalide

**Test1.5**: Nombre d'arguments insuffisant

**Test1.6**: Nombre d'arguments insuffisant

**Test1.7**: Option inconnue

**Test2**: int fileTest2.txt 1

**Test3**: int fileTest3.txt 1 1

**Test4**:

**Test5**: char fileTest4.txt 5

int fileTest4.txt 2

**Test6**: Erreur de syntaxe dans le fichier de mots-clés

**Test7**: Erreur lors de l'ouverture du fichier de mots-clés

**Test8**: Erreur lors de l'ouverture d'un fichier source

**Test9**: cout fileTest1.txt 3 4

endl fileTest1.txt 3 4

main fileTest1.txt 2 fileTest2.txt 1

return fileTest1.txt 5

**Test10**: cout fileTest1.txt 3 4

endl fileTest1.txt 3 4

main fileTest1.txt 2 fileTest2.txt 1

**Test11**: int fileTest1.txt 2 fileTest2.txt 1

**Test12**: int fileTest1.txt 2 fileTest2.txt 1

return fileTest1.txt 5

* Validation/commentaires éventuels

Tous les tests ont été validés.