

TP 3IF C++

Document de Réalisation

« Les Références Croisées »

Jan KEROMNES et David APARICIO

Sommaire

Réalisation de la classe	2
Choix de la structure pour les références croisées	2
Les occurrences, un vector<int>	2
Les fichiers, un vector<pair<string,X>>	2
Les identificateurs, une map<string,Y>	2
Structure finale	2
Listing des codes sources	2
Fichier « main.h »	2
Fichier « main.cpp »	3
Fichier « Stream.h »	7
Fichier « Stream.cpp »	8
Fichier « CrossReference.h »	11
Fichier « CrossReference.cpp »	12
Tests fonctionnels	14
Le script de test, « tests.bat »	14
Execution du script de test	15
Conclusion	16

Réalisation de la classe

Suite à la spécification complète de l'application, nous avons procédé à sa réalisation du TP.

Choix de la structure pour les références croisées

Pour représenter les « références croisées », nous avons eu une première approche théorique lors de la phase des spécifications. Nous avons ensuite appliqué ces choix à la réalisation de notre application C++, en utilisant la bibliothèque standard STL.

Les occurrences, un `vector<int>`

Pour représenter les numéros de lignes de nos identificateurs, nous avons choisi un `vector<int>`, auquel nous appliquons un `push_back(int)` à chaque nouveau numéro de ligne.

Les fichiers, un `vector<pair<string,X>>`

Pour représenter ces occurrences au sein des différents fichiers, nous avons premièrement défini une association entre le nom d'un fichier et ses occurrences avec une `pair<string, occurrences>`. Puis, nous avons créé une collection de ces paires, en utilisant un `vector<>`.

Les identificateurs, une `map<string,Y>`

Pour optimiser le processus de recherche, nous avions choisi un arbre rouge et noir. Cette structure se retrouve dans la STL sous la forme d'une `map<string,occurrencesFichiers>`.

Structure finale

Les trois points précédents nous donnent la structure suivante :

```
map<string, vector<pair<string, vector<int>>>>
```

Listing des codes sources

Fichier « main.h »

```
/*****************************************************************************
```

```

file name : main
started   : 10 nov. 2010
copyright : (C) 2010 by Jan and David (B3106)

*****
----- Interface of class <main> (file main.h)
#ifndef ! defined ( MAIN_H_ )
#define MAIN_H_


----- Types

// a structure for a reference to an identifier, in a given file at a given
line
struct REF
{
    string identifier;
    string file;
    int line;
};

----- Functions

int main ( int argc, char *argv[] );


#endif // MAIN_H_

```

Fichier « main.cpp »

```

*****
file name : main
started   : 10 nov. 2010
copyright : (C) 2010 by Jan and David (B3106)

*****
----- Realization of class <main> (file main.cpp)

//----- INCLUDES

//----- System Includes
using namespace std;
#include <iostream>
#include <fstream>
#include <string.h>
#include <map>
#include <set>
#include <vector>
#include <utility>

//----- Personal Includes
#include "main.h"
#include "CrossReferences.h"
#include "Stream.h"

```

```

//----- Constants
//----- PUBLIC
//----- Functions

void syntax ( string appName )
{
    cerr << "Call syntax: " << appName << " [-e] [-k keywordFile]"
[fileName]+" << endl;
}

void fileProblem ( string file )
{
    cerr << "Problem with " << file << endl;
}

int main ( int argc, char *argv[] )
{

    // if argument count is lower than 2, we have a syntax error
    if ( argc < 2 )
    {
        syntax( argv[0] );
        return -1;
    }

    CrossReferences references;
    bool keepKeywords = false;

    // do not handle argv[0] (the application's name)
    int handled = 1;

    // handle the "-e" flag
    if ( (string)argv[handled] == "-e" )
    {
        handled++; // we handled one more argument (-e flag)
        keepKeywords = true;
    }

    // handle the "-k" flag
    if ( (string)argv[handled] == "-k" )
    {
        handled+=2; // we handled two more arguments (-k flag and
keyword file name)

        // if the keyword file was not specified, we have a syntax
error
        if ( argc - handled <= 0 )
        {
            syntax( argv[0] );
            return -1;
        }
        else // we have the -k flag
        {

            #ifdef DEBUG

```

```

        cout << "Arguments : detected -k flag with file " <<
argv[handled-1] << endl;
#endif

        string file = argv[handled-1];

try
{
    // send the keywords to the class Stream
    Stream::ParseKeywordFile(file);
} catch (const char * error)
{
    fileProblem("'" + file + "' : " + error);
    return -1;
}
}

else // we do not have the -k flag
{

#ifdef DEBUG
cout << "Arguments : no -k flag detected, using C++ keywords"
<< endl;
#endif

// use the standard C++ keywords
vector<string> cpp;
cpp.push_back("asm");
cpp.push_back("auto");
cpp.push_back("bool");
cpp.push_back("break");
cpp.push_back("case");
cpp.push_back("catch");
cpp.push_back("char");
cpp.push_back("class");
cpp.push_back("const");
cpp.push_back("const_cast");
cpp.push_back("continue");
cpp.push_back("default");
cpp.push_back("delete");
cpp.push_back("do");
cpp.push_back("double");
cpp.push_back("dynamic_cast");
cpp.push_back("else");
cpp.push_back("enum");
cpp.push_back("explicit");
cpp.push_back("export");
cpp.push_back("extern");
cpp.push_back("false");
cpp.push_back("float");
cpp.push_back("for");
cpp.push_back("friend");
cpp.push_back("goto");
cpp.push_back("if");
cpp.push_back("inline");
cpp.push_back("int");
cpp.push_back("long");
cpp.push_back("mutable");
cpp.push_back("namespace");
}

```

```

    cpp.push_back("new");
    cpp.push_back("operator");
    cpp.push_back("private");
    cpp.push_back("protected");
    cpp.push_back("public");
    cpp.push_back("register");
    cpp.push_back("reinterpret_cast");
    cpp.push_back("return");
    cpp.push_back("short");
    cpp.push_back("signed");
    cpp.push_back("sizeof");
    cpp.push_back("static");
    cpp.push_back("static_cast");
    cpp.push_back("struct");
    cpp.push_back("switch");
    cpp.push_back("template");
    cpp.push_back("this");
    cpp.push_back("throw");
    cpp.push_back("true");
    cpp.push_back("try");
    cpp.push_back("typedef");
    cpp.push_back("typeid");
    cpp.push_back("typename");
    cpp.push_back("union");
    cpp.push_back("unsigned");
    cpp.push_back("using");
    cpp.push_back("virtual");
    cpp.push_back("void");
    cpp.push_back("volatile");
    cpp.push_back("wchar_t");
    cpp.push_back("while");

    for ( unsigned int i = 0 ; i < cpp.size() ; i++ )
    {
        Stream::AddKeyword(cpp[i]);
    }

}

// if there were no files to read, we have a syntax error
if ( argc - handled + 1 <= 1 )
{
    syntax( argv[0] );
    return -1;
}

// for each file to read
for(int i = 0 ; i < argc - handled ; i++)
{
    string file;
    try
    {
        // create a special identifier stream
        file = argv[i + handled];
        Stream stream(file, keepKeywords);

        // while the identifier stream has not reached the end of
the file
        while ( ! stream.EndOfFile() )
    }
}

```

```

    {
        // add the current identifier
        references.Add(stream.GetReference());
        // search for the next identifier
        stream.NextReference();
    }
} catch (const char * error)
{
    fileProblem("'" + file + "' : " + error);
    return -1;
}
}

references.Display();

return 0;
}

```

Fichier « Stream.h »

```

*****
file name : Stream
role      : An Identifier Stream parsing a file and searching for words
            that match or doesn't match a set of given keywords,
depending
            on specific flags.
started   : 17 nov. 2010
copyright : (C) 2010 by Jan and David (B3106)

*****
//----- Interface of class <Stream> (file Stream.h)
#ifndef ! defined ( STREAM_H_ )
#define STREAM_H_

//----- Used Interfaces
//----- Class <Stream>
//-----

class Stream
{

//----- PUBLIC
public:

//----- Constructors - Destructor
    Stream ( string file, bool keepKeywords );
    virtual ~Stream ( );

//----- Public Methods
    static void ParseKeywordFile(string file);
    static void AddKeyword(string keyword);
    REF GetReference(); // CONTRACT : Do not call if EndOfFile is
reached!

```

```

void NextReference();
bool EndOfFile();

//----- PROTECTED
protected:

//----- Protected Attributes
static set < string > keywords; // the keyword set
ifstream stream; // the stream reading the file
bool keep; // keep keywords or non-keywords
REF reference; // reference to the current Identifier
};

#endif // STREAM_H

```

Fichier « Stream.cpp »

```

*****file name : Stream
started : 5 déc. 2010
copyright : (C) 2010 by Jan and David (B3106)
*****/
```

----- Realization of class <Stream> (file Stream.cpp)

----- INCLUDES

----- System Includes

```

using namespace std;
#include <iostream>
#include <fstream>
#include <set>
#include <string.h>
```

----- Personal Includes

```

#include "main.h"
#include "Stream.h"
```

----- Constants

----- PUBLIC

----- Constructors - destructor

```

Stream::Stream ( string file, bool keepKeywords ) :
stream(file.c_str()), keep(keepKeywords)
{
    if ( ! stream.is_open() )
    {
        throw "File not found";
    }
    reference.file = file; // the file name
}
```

```

reference.line = 1; // start at line 1
reference.identifier = ""; // no Identifier found yet
NextReference(); // immediately search for the first Identifier

#ifdef DEBUG
cout << "<Stream> constructor call" << endl;
#endif
}

Stream::~Stream ( )
{
    #ifdef DEBUG
    cout << "<Stream> destructor call" << endl;
    #endif
}

//----- Public Methods

void Stream::ParseKeywordFile(string file)
{
    ifstream keywordStream(file.c_str());
    if ( !keywordStream )
    {
        throw "File not found";
    }

    // for each keyword in the file
    while ( keywordStream.good() )
    {
        string keyword;
        keywordStream >> keyword;
        if(keyword != "")
        {
            // add the keyword
            AddKeyword(keyword);
        }
    }
}

void Stream::AddKeyword(string keyword)
{
    // add keyword to list if it isn't already there
    if (keywords.find(keyword) == keywords.end())
keywords.insert(keyword);

    #ifdef DEBUG
    cout << "Added keyword : " << keyword << endl;
    #endif
}

REF Stream::GetReference()
{
    if (EndOfFile())
    {
}

```

```

        throw "ERROR: Impossible to retrieve reference if EndOfFile was
reached!";
    }

    // return the current reference to an Identifier (Identifier, file,
line)
    return reference;
}

void Stream::NextReference()
{
    #ifdef DEBUG
    cout << "call to NextReference()" << endl;
    #endif

    char c;
    string word;
    bool found = false;

    while ( !found && !stream.eof() )
    {
        word = "";

        stream.get(c);
        if (c == '\n')
        {
            // if we reach a new line, increment the line counter
            reference.line++;
        }

        // if we have an alphabetic character
        if( isalpha(c) )
        {
            while ( isalpha(c) || isdigit(c) || c == '_' )
            {
                // add it and all the following alpha, digits and
'-'s to the word
                word.push_back(c);
                stream.get(c);
            }

            // once the word is complete, test if it is an identifier
            // i.e. a keyword if we do keep the keywords, or a non-
keywords otherwise
            if ( ( keep && ( keywords.find(word) !=
keywords.end() ) ) ||
                  ( !keep && ( keywords.find(word) ==
keywords.end() ) ) )
            {
                #ifdef DEBUG
                cout << "Found Identifier : " << word << endl;
                #endif

                // we have found a new Identifier at the current
file name and line, stop here
                found = true;
                reference.identifier = word;
            }
        }
    }
}

```

```

        }
    }

bool Stream::EndOfFile()
{
    return stream.eof();
}

// declaration of the static keyword set for visibility reasons
set < string > Stream::keywords;

```

Fichier « CrossReference.h »

```

*****  

file name : CrossReferences  

role      : Used to store cross-references to identifiers in several  

           files at several lines.  

started   : 17 nov. 2010  

copyright : (C) 2010 by Jan and David (B3106)  

*****  

  

//----- Interface of class <CrossReferences> (file CrossReferences.h)
#ifndef CROSSREFERENCES_H_
#define CROSSREFERENCES_H_  

  

//----- Used Interfaces  

  

//----- Constants  

  

//----- Types  

  

// Cross-reference structure decomposition:  

/* CROSS_REF  := map < IDENT_NAME , FILE_VECT >  

 * IDENT_NAME := string  

 * FILE_VECT   := vector < REF_PAIR >  

 * REF_PAIR    := pair < FILE_NAME, REF_VECT >  

 * FILE_NAME   := string  

 * REF_VECT    := vector < LINE_NUM >  

 * LINE_NUM    := int */  

  

typedef vector < pair < string , vector < int > > > FILE_VECT;  

typedef map < string , FILE_VECT > CROSS_REF;  

  

//-----  

// Class <CrossReferences>  

//-----  

  

class CrossReferences
{
  

//----- PUBLIC
public:
  

//----- Constructors - destructor

```

```

CrossReferences ( );
virtual ~CrossReferences ( );

//----- Public Methods
void Add (REF ref); // add a new reference (for identifier I, in file
F, at line L)
void Display(); // display the collected cross-references in the
standard output

//----- PROTECTED
protected:

//----- Protected Attributes
CROSS_REF references;

};

#endif // CROSSREFERENCES_H_

```

Fichier « CrossReference.cpp »

```

*****
file name : CrossReferences
started   : 17 nov. 2010
copyright : (C) 2010 by Jan and David (B3106)
*****

//----- Realization of class <CrossReferences> (file CrossReferences.cpp)

//----- INCLUDES

//----- System Includes
using namespace std;
#include <iostream>
#include <map>
#include <vector>
#include <utility>

//----- Personal Includes
#include "main.h"
#include "CrossReferences.h"

//----- Constants

//----- PUBLIC

//----- Constructors - Destructor

CrossReferences::CrossReferences ( ) : references()
{
    #ifdef DEBUG
        cout << "<CrossReferences> constructor call" << endl;
    #endif
}

```

```

}

CrossReferences::~CrossReferences ( )
{

    #ifdef DEBUG
        cout << "<CrossReferences> destructor call" << endl;
    #endif
}

//----- Public Methods

void CrossReferences::Add (REF ref)
{
    #ifdef DEBUG
        cout << "Adding reference : " << ref.identifier << " " << ref.file <<
" " << ref.line << endl;
    #endif

    // if the identifier doesn't exist, or if the identifier's last file
    // doesn't match the reference file,
    if (references.find(ref.identifier) == references.end() ||
references[ref.identifier].back().first != ref.file)
    {
        // add the file to the identifier (file name and new
vector<int> for the line numbers)
        references[ref.identifier].push_back(make_pair(ref.file,
vector<int>()));
    }
    // add the line number of the reference to the corresponding
identifier and file
    references[ref.identifier].back().second.push_back(ref.line);
}

void CrossReferences::Display()
{
    // for each identifier
    for ( CROSS_REF::iterator i = references.begin(), end =
references.end() ; i != end ; i++ )
    {
        // print its name
        cout << i->first;
        FILE_VECT fileVector = i->second;
        // for each file
        for ( int i = 0, end = fileVector.size() ; i < end ; i++ )
        {
            // print its name
            cout << '\t' << fileVector[i].first;
            vector < int > lineVector = fileVector[i].second;
            // for each line number
            for ( int i = 0, end = lineVector.size() ; i < end ;
i++ )
            {
                // print the line number
                cout << ' ' << lineVector[i];
            }
        }
        cout << endl;
}

```

```
}
```

Tests fonctionnels

Le script de test, « tests.bat »

```
::Test 1  
TP_STL.exe  
  
::Test 2  
TP_STL.exe unexistant_file  
  
::Test 3  
TP_STL.exe file1.cpp file1.h  
  
::Test 4  
TP_STL.exe -e file1.cpp file1.h  
  
::Test 5  
TP_STL.exe -k keywords.txt file1.cpp file1.h  
  
::Test 6  
TP_STL.exe -e -k keywords.txt file1.cpp file1.h  
  
::Test 7  
TP_STL.exe -e -k nokeywords.txt file1.cpp file1.h  
  
pause
```

Dans ce **script batch pour Windows**, nous effectuons tous les tests décrits dans le document de spécifications, afin de vérifier que les résultats obtenus correspondent bien à ce qu'on attendait.

Execution du script de test

```
C:\Windows\system32\cmd.exe

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe
Call syntax: TP_STL.exe [-e] [-k keywordFile] [fileName]+

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe unexistant_file
Problem with 'unexistant_file' : File not found

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe file1.cpp file1.h
Hello    file1.cpp 1 4
affiche  file1.cpp 1
cout     file1.cpp 4
endl     file1.cpp 4
le       file1.cpp 1
main    file1.cpp 2      file1.h 1
message  file1.cpp 1
world   file1.cpp 1 4

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe -e file1.cpp file1.h
int     file1.cpp 2      file1.h 1
return  file1.cpp 5

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe -k keywords.txt file1.cpp file1.h
Hello    file1.cpp 1 4
affiche  file1.cpp 1
cout     file1.cpp 4
endl     file1.cpp 4
le       file1.cpp 1
main    file1.cpp 2      file1.h 1
message  file1.cpp 1
return  file1.cpp 5

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe -e -k keywords.txt file1.cpp file1.h
int     file1.cpp 2      file1.h 1
world  file1.cpp 1 4

C:\eclipse\workspace\TP_STL\Debug>TP_STL.exe -e -k nokeywords.txt file1.cpp file1.h
Appuyez sur une touche pour continuer...
```

Comme nous pouvons le voir ici, les résultats de tous les tests correspondent aux résultats prévus par les spécifications : **l'application est donc validée.**

Conclusion

Pourquoi pouvons-nous dire que l'application du TP « Références Croisées » a été réalisée avec succès ?

Tout d'abord, nous avons analysé les missions qu'elle devait remplir, et étudié les structures de données s'appliquant au mieux à notre besoin très spécifique : celui de pouvoir représenter des « références croisées ».

Une fois cette phase de réflexions menée à bien, tous les éléments de l'application ont été spécifiés de manière précise, suite à quoi nous avons pu concevoir et réaliser l'application.

Le programme a ensuite été validé par une politique de tests méthodiques et exhaustifs. Nous avons ainsi pu vérifier que l'application correspondait bien aux spécifications, et que les besoins du client ont bien tous été satisfaits.

L'application peut donc être livrée dans les meilleures conditions.