Main.cpp

```
/*************************************************************************
TPLog  -  fichier main
-------------------
début                : 23/11/2015 14:30:19
copyright            : (C) 2015 par Quentin SCHROTER, Nicolas GRIPONT
e-mail               : quentin.schroter@insa-lyon.fr , nicolas.gripont@insa-lyon.fr
*************************************************************************/

#include <string>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <string.h>

using namespace std;

#include "GraphDocuments.h"
#include "LogParser.h"

#ifdef TEST

#include "TestLog.h"

int main()
// Algorithme
//
{
   test_LogParser_Parser();
   return 0;
}//----- Fin de Main

#else

static const int NB_DOC = 10; // nombre de document a afficher pour la commande par defaut

void defaut (GraphDocuments & graph, int n , bool uniquementReussis = true );
// Mode d'emploi :
// Fonction permettant d'afficher les NB_DOC les plus consultes d'un GraphDocuments
//
// Contrat : aucun.
//

void creerFichierGraphe (const GraphDocuments & graph, const string & nomFichierGraphe , bool
uniquementReussis = true );
// Mode d'emploi :
// Fonction permettant de creer un fichier a partir GraphViz d'un GraphDocuments
//
// Contrat : aucun.
//

void initialiserGraphe (GraphDocuments & graph, const string & fichierLog, bool exclusion, int
heure );
```

```
// Mode d'emploi :
// Fonction permettant d'initialiser un Graphdocuments a partir
// d'un fichier.
//
// Contrat : aucun.
//

bool exclure ( const LogLine & l , const vector<string> & v );
// Mode d'emploi :
// Fonction permettant d'analyser si le fichier consultr dans une ligne de log
// et a exclure
//
// Contrat : aucun.
//

vector<string> extensionsExclus ( );
// Mode d'emploi :
// Fonction retournant un vecteur contenant les extensions des fichiers
// a exclure.
//
// Contrat : aucun.
//


//faire une fonction man de ce programme si erreur commande


int main (int argc, char* argv[])
{
    string nomFichierLog = "";
    string nomFichierGraph = "";
    string serverAdress = "";
    int heure = -1;
    bool exclusion = false;

//    cout << "argc = " << argc << endl;

    if ( argc < 3 )
    {
        cerr << "Commande invalide." << endl;
        exit(1);
    }
    else
    {
        nomFichierLog = argv[argc-2];
        serverAdress = argv[argc-1];
    }

    for (int i = 1; i < argc - 2; i++){
        if (strcmp(argv[i],"-g")==0)
        {
            i++;
            if ( i < argc - 2 )
            {
                nomFichierGraph = argv[i];
            }
```

```
            else
            {
                cerr << "Commande invalide." << endl;
                exit(1);
            }
        }
        else if (strcmp(argv[i],"-e")==0)
        {
            exclusion = true;
        }
        else if (strcmp(argv[i],"-t")==0)
        {
            i++;
            if ( i < argc - 2 )
            {
                if ( sscanf(argv[i],"%d",&heure) != 1 && heure < 0 && heure > 24 )
                {
                    cerr << "Commande invalide." << endl;
                    exit(1);
                }
            }
            else
            {
                cerr << "Commande invalide." << endl;
                exit(1);
            }
        }
        else
        {
            cerr << "Commande invalide." << endl;
            exit(1);
        }
    }


//    cout << "nomFichierLog : " << nomFichierLog << endl;
//    cout << "nomFichierGraph : " << nomFichierGraph << endl;
//    cout << "serverAdress : " << serverAdress << endl;
//    cout << "heure : " << heure << endl;
//    cout << "exclusion : " << exclusion << endl;

    GraphDocuments graph(serverAdress);

    initialiserGraphe(graph,nomFichierLog,exclusion,heure);

    if ( nomFichierGraph != "")
    {
        creerFichierGraphe(graph, nomFichierGraph);
    }

    defaut (graph, NB_DOC);

//    int somme = 0;

//    vector<Document*> v = graph.Documents();
//    for ( vector<Document*>::const_iterator itv = v.begin(); itv != v.end(); itv++ )
```

```
//    {
//        somme += (*itv)->NbHits().NombreDeHitsTotal(false);
//    }
//    cout << somme << endl;

          return 0;
}


void defaut ( GraphDocuments & graph , int n, bool uniquementReussis )
{
   int i = 0;
   graph.TrierParNombreDeHitsReussis();
   const vector<Document*> documents = graph.Documents();
   vector<Document*>::const_iterator itv = documents.begin();

   while ( itv != documents.end() && i < n )
   {
      cout << (*itv)->CheminAccesRessource() << " ("
           << (*itv)->NbHits().NombreDeHitsTotal(uniquementReussis) <<
             " hits)" << endl;
      itv++;
      i++;
   }
}

void creerFichierGraphe ( const GraphDocuments & graph, const string & nomFichierGraphe , bool
uniquementReussis )
{
   ofstream fichier(nomFichierGraphe,ios::out | ios::trunc);

   if(fichier)
   {
      fichier << "digraph {" << endl;

      const vector<Document*> documents = graph.Documents();
      int i = 0;
      map<Document*,int> positions;

      for ( vector<Document*>::const_iterator itv = documents.begin(); itv != documents.end(); itv++
)
      {
         if ( (*itv)->NombreDeHitsAPartirDeCeDocument(uniquementReussis) > 0 || (*itv)-
>NbHits().NombreDeHitsTotal(uniquementReussis) > 0 )
         {
            fichier << "node" << i << "[label=\"" << (*itv)->CheminAccesRessource() << "\"];" << endl;
            positions.insert(make_pair((*itv),i));
            i++;
         }
      }

      for ( vector<Document*>::const_iterator itv = documents.begin(); itv != documents.end(); itv++
)
      {
         for ( map<string,pair<Document*, NombreDeHits>>::const_iterator itm = (*itv)-
>DocumentsAtteignables().begin(); itm != (*itv)->DocumentsAtteignables().end(); itm++ )
```

```
            {
                if ( itm->second.second.NombreDeHitsTotal(uniquementReussis) > 0 )
                {
                    fichier << "node" << positions[(*itv)] << " -> " <<
                        "node" << positions[itm->second.first] << " [label=\"" <<
                        itm->second.second.NombreDeHitsTotal(uniquementReussis) << "\"];" << endl;
                }
            }
        }
        fichier << "}" << endl;
        fichier.close();
    }
    else
    {
        cerr << "Impossible d'ouvrir " << nomFichierGraphe << " !" <<endl;
    }
}


void initialiserGraphe (GraphDocuments & graph, const string & fichierLog, bool exclusion, int
heure )
{
    ifstream fichier(fichierLog,ios::in);
    vector<string> v = extensionsExclus();
    if(fichier)
    {
        string logLine;

        while ( getline(fichier, logLine) )
        {
            LogLine l = LogParser::Parser(logLine,graph.NomDomaine());
            if ( !exclusion && heure == -1 )
            {
                graph.TraiterLogLine(l);
            }
            else if ( !exclusion && l.date.heure == heure  )
            {
                graph.TraiterLogLine(l);
            }
            else if ( exclusion && !exclure(l,v) && heure == -1 )
            {
                graph.TraiterLogLine(l);
            }
            else if ( exclusion && !exclure(l,v) && l.date.heure == heure )
            {
                graph.TraiterLogLine(l);
            }
        }
        fichier.close();
    }
    else
    {
        cerr << "Impossible d'ouvrir " << fichierLog << " !" <<endl;
    }
}
```

```
bool exclure ( const LogLine & l , const vector<string> & v )
// Algorithme :
//
{
    string extension = LogParser::Extension(l.requestedURL);
    for ( vector<string>::const_iterator it = v.begin(); it != v.end(); it++ )
    {
        if ( *it == extension )
        {
            return true;
        }
    }
    return false;
} //----- Fin de Exclure

vector<string> extensionsExclus ( )
// Algorithme :
//
{
    vector<string> v;
    v.push_back(".css");
    v.push_back(".jpg");
    v.push_back(".jpeg");
    v.push_back(".jpe");
    v.push_back(".gif");
    v.push_back(".png");
    v.push_back(".ico");
    v.push_back(".js");
    return v;
} //----- Fin de extensionsExclus


#endif
```