```cpp
/*************************************************************************
                          Graphe  -  description
                            -------------------
    debut                : 23/11/2015
    copyright            : (C) 2015 par mfallouh_mvirsolvy
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
using namespace std;
#include <iostream>
#include <string>
#include<fstream>
//------------------------------------------------------ Include personnel
#include "Graphe.h"

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

extern const string EXCLUSIE[];
extern const int NB_EXTENSIONS;
extern const char SEP_PT;
//----------------------------------------------------- Variables de classe

//-------------------------------------------------------------- Types prives


//----------------------------------------------------------------- PUBLIC
//-------------------------------------------------------- Fonctions amies

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

//------------------------------------------------ Surcharge d'operateurs

/*
Graphe & Graphe::operator = ( const Graphe & unGraphe )
// Algorithme :
//
{
} //----- Fin de operator =
*/

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

/*
Graphe::Graphe ( const Graphe & unGraphe )
// Algorithme :
//
{
#ifdef MAP
    cout << "Appel au constructeur de copie de <Graphe>" << endl;
#endif
} //----- Fin de Graphe (constructeur de copie)
*/

bool Graphe::EstImage(const string & adresse)
```

```
// Algorithme :
//
{
	size_t posExtension = adresse.find_last_of(SEP_PT); // position à partir de
laquelle commence l'extension
	string Extension = adresse.substr(posExtension + 1,adresse.npos); //
l'extension
	bool image = false; // s'il s'agit d'une image
	int i = 0;
	while( i < NB_EXTENSIONS && !image ) // parcours les extensions images.
	{
		image = Extension.compare(EXCLUSIE[i]) == 0;// si l'extension est
celle d'une image
		i++;
	} // fin du parcours
	return image;
} // ------ Fin de estImage

void Graphe::GenereFichier(const string & nomFichier)
// Algorithme :
// Parcours la map des noeuds en les affichant tous, selon la synthaxe,
// de même pour les liens.
{
	ofstream grapheFile(nomFichier.c_str());
	if (grapheFile) // le fichier est correct
	{
		grapheFile << "digraph {" << endl;

		/*generation des noeuds */
		map<string, int > ::const_iterator noeudsDebut, noeudsFin;
		noeudsDebut = noeuds.begin();
		noeudsFin = noeuds.end();
		for (noeudsDebut; noeudsDebut != noeudsFin; noeudsDebut++) //
parcours de noeuds
		{
			grapheFile << "node" << noeudsDebut->second << " [" <<
noeudsDebut->first << "];" << endl;
		}

		/*generation des liens */
		map<paire, int > ::const_iterator liensDebut, liensFin;
		liensDebut = liens.begin();
		liensFin = liens.end();
		for (liensDebut; liensDebut != liensFin; liensDebut++) // parcours de
liens
		{
			grapheFile << "node" << liensDebut->first.NumReferer << " ->
node" << liensDebut->first.NumCible << " [label=\"" << liensDebut->second <<
"\"];" << endl;
		}

		grapheFile << "}" << endl;
	} // fin de fichier correct
	else
	{
		cerr << "[Generation Graphe] Pb sur le fichier" << endl;
	}
```

```
} // ----- Fin de GenereFichier

Graphe::Graphe ( const Collection &aCol, const bool e , const int t    ) :
       valeurNoeud(0)
// Algorithme :
// parcours la collection dans son ensemble,
// verifie que la cible est en accord avec l'option e, puis l'insere dans la
graphe en fonction
// avec la methode genere graphe
{
#ifdef MAP
    cout << "Appel au constructeur de <Graphe>" << endl;
#endif
       map<string , Cible > :: const_iterator debut,fin; // les iterateurs de
parcours de la collection
       debut=aCol.pages.begin();
       fin=aCol.pages.end();

       for(debut ; debut!=fin; debut++) // iteration de parcours de la map pages
       {
              bool estImage = e && EstImage(debut->first); // permet de gerer
l'option -e

              if (!(e && EstImage(debut->first))) // si la cible peut être ajoutee
dans le graphe
              {
                     if (t != -1) // filtre en fonction de l'heure
                     {
                            creeGrapheHeure(debut, t, e);
                     }
                     else if (t == -1)
                     {
                            for (size_t heure = 0; heure < 24; heure++) // on le
fait pour chaques heures !!!
                            {
                                   creeGrapheHeure(debut, heure, e);
                            }
                     }// fin du else if
              } // fin de la creation de la cible dans le graphe

       } // fin du parcours de la Collection

} //----- Fin de Graphe


Graphe::~Graphe ( )
// Algorithme :
//
{
#ifdef MAP
    cout << "Appel au destructeur de <Graphe>" << endl;
#endif
} //----- Fin de ~Graphe

void Graphe::creeGrapheHeure(map<string, Cible>::const_iterator & cible, const
size_t & heure, bool e)
// Algorithme :
```

```
// parcours la cible, pour seleectionner les GET.
// parcours ensuite les logs, et met à jour noeuds et liens en fonction du referer
du log, et des options.
// si le referer est en accords avec les options alors il cree dans noeuds si
besoin, puis le liens entre
// la cible et le referer est cree ou incremente.
{
	bool noeudCree = false; // savoir si on a cree un noeud pour la cible ou
pas encore
							// permettra de creer le noeud
uniquement si il interagit avec une autre page

	int valeurNoeudCible;	// garder en memoire la valeur du noeud cible

	// declaration des iterateurs de parcours de la cible
	map<string, list<Log> > ::const_iterator typeReqDeb, typeReqFin;
	if (!cible->second.lesLogs[heure].empty())// si la liste est non vide
	{
		typeReqDeb = cible->second.lesLogs[heure].begin();
		typeReqFin = cible->second.lesLogs[heure].end();


		while (typeReqDeb != typeReqFin && typeReqDeb->first != "GET")//
isolation des hits
		{
			typeReqDeb++;
		}

		if (typeReqDeb != typeReqFin && typeReqDeb->first == "GET") // si la
page a bien ete hit au moins une fois.
		{

			list<Log> ::const_iterator cur = typeReqDeb->second.begin();
// iterateur de parcours des logs

			/*parcours des logs*/
			while (cur != typeReqDeb->second.end())
			{

				if (!e || (e && !EstImage(cur->referer))) // si
l'extension est OK
				{
					if (!noeudCree) // si on a pas encore eu besoin
de creer le noeud de la cible
					{
						if (creeNoeud(cible->first, valeurNoeud))
// si le noeud doit etre cree
						{
							valeurNoeudCible = valeurNoeud;
							valeurNoeud++;
						}
						else // si il etait deja present
						{
							valeurNoeudCible =
noeuds.find(cible->first)->second;
						}
					} // fin du !noeudCree
```

```
                                    /*Mise a jour des noeuds*/
                                    if (creeNoeud(cur->referer, valeurNoeud)) // si
le noeud du referer n'est pas deja present
                                    {
                                          valeurNoeud++;
                                    }

                                    /*Mise à jour des liens*/
                                    paire paireInserer = { noeuds.find(cur-
>referer)->second , valeurNoeudCible };
                                    pair<paire, int> insertion = { paireInserer, 1
};
                                    pair<map<paire,int >::iterator, bool>
bInsertion;
                                    bInsertion = liens.insert(insertion);
                                    if (!bInsertion.second)
                                    {
                                          liens.find(paireInserer)->second++;
                                    }
                              } // fin de extension OK

                              cur++;
                        }
                  } // fin du si page hit
            } // fin du si liste non vide
} // ------ Fin de creeGrapheHeure

bool Graphe::creeNoeud(const string & page, const int & valeurNoeud)
// Algorithme :
//
{
      pair<string, int> aInserer = { page, valeurNoeud }; // le noeud
      pair<map<string, int >::iterator, bool> retInsertion;
      retInsertion = noeuds.insert(aInserer);
      return retInsertion.second;
} // ------ Fin de creeNoeud


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

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

//-------------------------------------------------------- Methodes privees
```