Compte-rendu du TP C++ 3
Code source

Marc Gagné Selma Nemmaoui

Décembre 2015

```
/*************************************************************************
                ConfigReader - Lecteur de fichier de configuration
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <cstdlib>
#include <fstream>
#include <stdexcept>
#include <sstream>

//------------------------------------------------------ Include personnel
#include "ConfigReader.h"
#include "Logger.h"

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

//----------------------------------------------------- Méthodes publiques
int ConfigReader::GetInteger (const std::string & key, int def) const
// Algorithme : Trouve la valeur désirée si elle existe, puis la convertit en
// entier.
{
    try
    {
        return atoi(config.at(key).c_str());

    }
    catch (std::out_of_range & e)
    {
        return def;
    }
}

std::unordered_set<std::string> ConfigReader::GetSet (
        const std::string & key, const std::unordered_set<std::string> & def
) const
// Algorithme : Trouve la valeur désirée si elle existe, puis sépare la valeur
// stockée selon les virgules.
{
    try
    {
        std::stringstream ss(config.at(key));
        std::unordered_set<std::string> elems;
        std::string item;
        while (std::getline(ss, item, ','))
        {
            elems.insert(item);
        }
        return elems;
    }
    catch (std::out_of_range & e)
    {
        return def;
    }
}
```

```
std::string ConfigReader::GetString (
        const std::string & key, const std::string & def
) const
{
    try
    {
        return config.at(key);
    }
    catch (std::out_of_range & e)
    {
        return def;
    }
}

//------------------------------------------------------ Constructeurs - destructeur
ConfigReader::ConfigReader (const std::string & filename)
// Algorithme : Lit un fichier de configuration ligne par ligne et en extrait
// la clé et la valeur, puis ferme le flux de fichier.
{
    DEBUG("Appel au constructeur de ConfigReader");
    std::ifstream file(filename);
    std::string line;

    if (!file.is_open())
    {
        return;
    }

    while (!std::getline(file, line).eof())
    {
        if (line.empty())
        {
            continue;
        }
        std::string param;
        std::string value;

        unsigned long posEqual = line.find('=');
        param = line.substr(0, posEqual);
        value = line.substr(posEqual + 1);
        config[param] = value;
    }

    file.close();

} //----- Fin du constructeur

ConfigReader::~ConfigReader ()
{
    DEBUG("Appel au destructeur de ConfigReader");
} //----- Fin du destructeur
```

```
/*************************************************************************
            ConfigReader - Lecteur de fichier de configuration
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

// Interface de la classe <ConfigReader> (fichier ConfigReader.h)
#if ! defined ( CONFIG_READER_H )
#define CONFIG_READER_H

//------------------------------------------------------------ Interfaces utilisées
#include <string>
#include <unordered_map>
#include <unordered_set>

//------------------------------------------------------------------------------
// Rôle de la classe <ConfigReader>
// Lit un fichier de configuration et permet d'en extraire les associations
// clés -> valeurs. Un fichier de configuration est structuré de la manière
// suivante :
// cle1=valeur1
// cle2=valeur2
//------------------------------------------------------------------------------

class ConfigReader
{
//----------------------------------------------------------------- PUBLIC
public:
//---------------------------------------------------------- Méthodes publiques
    int GetInteger (const std::string & key, int def = 0) const;
    // <key> : la clé associée à la valeur désirée
    // <def> : la valeur par défaut à utiliser
    // Mode d'emploi : Essaie de trouver la clé <key>. Si elle existe, renvoie
    // la valeur associée convertie en entier ; sinon, renvoie la valeur par
    // défaut <def>.

    std::unordered_set<std::string> GetSet (
            const std::string & key, const std::unordered_set<std::string> & def
    ) const;
    // <key> : la clé associée à la valeur désirée
    // <def> : la valeur par défaut à utiliser
    // Mode d'emploi : Essaie de trouver la clé <key>. Si elle existe, renvoie
    // la valeur associée convertie en set ; sinon, renvoie la valeur par défaut
    // <def>.

    std::string GetString (const std::string & key, const std::string & def = ""
    ) const;
    // <key> : la clé associée à la valeur désirée
    // <def> : la valeur par défaut à utiliser
    // Mode d'emploi : Essaie de trouver la clé <key>. Si elle existe, renvoie
    // la valeur associée ; sinon, renvoie la valeur par défaut <def>.

//------------------------------------------------ Constructeurs - destructeur
    ConfigReader (const std::string & filename);
    // <filename> : le nom du fichier de configuration à lire
    // Mode d'emploi : Initialise le lecteur à partir du fichier de
    // configuration spécifié par <filename>.

    virtual ~ConfigReader ();
    // Mode d'emploi : Détruit le lecteur.
```

```
//---------------------------------------------------------------------------- PRIVE
protected:
//---------------------------------------------------------------------- Attributs protégés
    std::unordered_map<std::string, std::string> config;
};

#endif // CONFIG_READER_H
```

```
/*************************************************************************
                Document - Représente un document sur le serveur
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include personnel
#include "Document.h"
#include "Logger.h"

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

//----------------------------------------------------- Méthodes publiques
void Document::AddLocalHit ()
{
    DEBUG("Appel à Document::AddLocalHit");
    localHits++;
} //----- Fin de AddLocalHit

void Document::AddRemoteHit (unsigned long documentId)
// Algorithme : Crée une case pour le document si elle n'existe pas déjà, puis
// incrémente le compteur dans cette case de 1.
{
    DEBUG("Appel à Document::AddRemoteHit");
    if (!remoteHits.count(documentId))
    {
        remoteHits[documentId] = 0;
    }
    remoteHits[documentId]++;
} //----- Fin de AddRemoteHit

unsigned int Document::GetLocalHits () const
{
    DEBUG("Appel à Document::GetLocalHits");
    return localHits;
} //----- Fin de GetLocalHits

const DocumentHits & Document::GetRemoteHits () const
{
    DEBUG("Appel à Document::GetRemoteHits");
    return remoteHits;
} //----- Fin de GetRemoteHits

const std::string & Document::GetUri () const
{
    DEBUG("Appel à Document::GetUri");
    return uri;
} //----- Fin de GetUri

//-------------------------------------------------- Surcharge d'opérateurs
bool Document::operator > (const Document & document) const
{
    DEBUG("Appel à Document::operator >");
    return localHits > document.localHits;
} //----- Fin de la surcharge d'opérateur >
```

```
//-------------------------------------------------- Constructeurs - destructeur
Document::Document (const std::string & documentUri) :
        localHits(0), uri(documentUri)
{
    DEBUG("Appel au constructeur de Document");
} //----- Fin du constructeur

Document::~Document()
{
    DEBUG("Appel au destructeur de Document");
} //----- Fin du destructeur
```

```
/**************************************************************************
                Document - Représente un document sur le serveur
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
**************************************************************************/

// Interface de la classe <Document> (fichier Document.h)
#if ! defined ( DOCUMENT_H )
#define DOCUMENT_H

//------------------------------------------------------------ Interfaces utilisées
#include <string>
#include <unordered_map>

//------------------------------------------------------------------------ Types
typedef std::unordered_map<unsigned long, unsigned int> DocumentHits;

//------------------------------------------------------------------------------
// Rôle de la classe <Document>
// Un <Document> représente un document sur le serveur local. La classe stocke
// le nombre de fois que le document a été atteint, soit à partir d'un autre
// document sur le serveur loca, soit à partir d'un site extérieur ou inconnu.
// Note : il existe un document spécial (avec pour URI "*") qui représente
// l'extérieur. Celui-ci ne devrait jamais être atteint, et peut seulement être
// une source d'accès au serveur local.
//------------------------------------------------------------------------------

class Document
{
//------------------------------------------------------------------------ PUBLIC
public:
//----------------------------------------------------------- Méthodes publiques
    void AddLocalHit ();
    // Mode d'emploi : Incrémente le nombre de fois que ce document a été
    // atteint par d'autres documents, ce qui inclut des documents externes.

    void AddRemoteHit (unsigned long documentId);
    // <documentId> : l'ID du document que ce document a atteint
    // Mode d'emploi : Incrémente le nombre de fois que le document <documentId>
    // a été atteint par ce document. Si le document <documentId> n'avait jamais
    // été atteint, le compte est initialisé à 1.
    // Contrat : <documentId> doit être un index dans le vecteur <documents> de
    // HistoryManager.

    unsigned int GetLocalHits () const;
    // Mode d'emploi : Renvoie le nombre de fois que ce document a été atteint.

    const DocumentHits & GetRemoteHits () const;
    // Mode d'emploi : Renvoie une liste associative de documents (identifiés
    // par leur ID) et du nombre de fois qu'ils ont été atteints par celui-ci.
    // Si un document n'est pas dans la liste, il n'a jamais été atteint par ce
    // document.

    const std::string & GetUri () const;
    // Mode d'emploi : Renvoie l'URI correspondant au document. À noter que les
    // paramètres (énumérés après l'extension, généralement en commençant par un
    // ?) ne sont pas pris en compte : index.php est le même document que
    // index.php?p=1. Si l'URI renvoyée est égale à "*", le "document" n'est pas
    // un vrai document, mais représente tous les documents externes, qui ne
```

```
    // sont pas sur le serveur local.

//------------------------------------------------------- Surcharge d'opérateurs
    bool operator > (const Document & document) const;
    // <document> : le document auquel comparer ce document
    // Mode d'emploi : Renvoie vrai si ce document a été atteint plus de fois
    // que l'autre document, et faux sinon.

//-------------------------------------------------- Constructeurs - destructeur
    Document (const std::string & documentUri);
    // <documentUri> : l'URI identifiant ce document (sans paramètres)
    // Mode d'emploi : Crée un nouveau document à partir de son URI,
    // préalablement épurée de ses paramètres (les informations variables qui
    // peuvent suivre l'extension de fichier, généralement précédées d'un ?).

    virtual ~Document ();
    // Mode d'emploi : détruit le document.

//------------------------------------------------------------------------ PRIVE
protected:
//----------------------------------------------------- Attributs protégés
    unsigned int localHits;  // Le nombre de fois que le document a été atteint
    DocumentHits remoteHits;  // Le nombre de fois que d'autres documents
    std::string uri;  // L'URI de ce document, relatif au serveur local
};

#endif // DOCUMENT_H
```

```
/*************************************************************************
                    DotFileWriter - Écrivain de DOT-file
                    -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include personnel
#include "DotFileWriter.h"
#include "Logger.h"

//------------------------------------------------------------ Constantes
const std::string DotFileWriter::HEADER = "digraph {\n";
const std::string DotFileWriter::FOOTER = "}\n";

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

//-------------------------------------------------------- Méthodes publiques
void DotFileWriter::AddLink (
        unsigned long sourceId, unsigned long targetId,
        const std::string & linkLabel
)
{
    DEBUG("Appel à DotFileWriter::AddLink");
    links.push_front({sourceId, targetId, linkLabel});
} //----- Fin de AddLink

void DotFileWriter::AddNode (unsigned long id, const std::string & label)
{
    DEBUG("Appel à DotFileWriter::AddNode");
    nodes[id] = label;
} //----- Fin de AddNode

void DotFileWriter::Close ()
{
    DEBUG("Appel à DotFileWriter::Close");
    if (dotFile.is_open())
    {
        dotFile.close();
    }
} //----- Fin de Close

void DotFileWriter::InitGraph (unsigned long graphNodes)
// Algorithme : Supprime tout ancien graphe qui avait été créé en détruisant
// tous ses noeuds et en réinitialisant la liste de liens, puis prépare un
// nouveau tableau de noeuds.
{
    DEBUG("Appel à DotFileWriter::InitGraph");
    if (nodes != nullptr)
    {
        links.clear();
        delete[] nodes;
    }
    numNodes = graphNodes;
    nodes = new std::string[numNodes];
} //----- Fin de InitGraph
```

```
bool DotFileWriter::Open (const std::string & filename)
// Algorithme : Ferme tout flux déjà ouvert et ouvre un nouveau lié au fichier
// <filename>.
{
    DEBUG("Appel à DotFileWriter::Open");
    Close();
    dotFile.open(filename);
    return dotFile.is_open();
} //----- Fin de Open

void DotFileWriter::Write ()
// Algorithme : Génère le fichier DOT en listant tous les noeuds d'abord, suivis
// de tous les liens.
{
    DEBUG("Appel à DotFileWriter::Write");
    if (!dotFile.is_open() || nodes == nullptr)
    {
        ERROR("DOT-file not opened or graph not initialized");
        return;
    }
    dotFile << HEADER;
    for (unsigned int i = 0; i < numNodes; i++)
    {
        dotFile << "node" << i << " [label=\"" << nodes[i] << "\"];\n";
    }
    for (Link & link : links)
    {
        dotFile << "node" << link.sourceNodeId << " -> node" <<
                link.destinationNodeId << " [label=\"" << link.label <<
                "\"];\n";
    }
    dotFile << FOOTER;
} //----- Fin de Write

//-------------------------------------------------- Constructeurs - destructeur
DotFileWriter::DotFileWriter () : nodes(nullptr), numNodes(0)
{
    DEBUG("Appel au constructeur de DotFileWriter");
} //----- Fin du constructeur

DotFileWriter::~DotFileWriter ()
{
    DEBUG("Appel au destructeur de DotFileWriter");
    if (nodes != nullptr)
    {
        delete[] nodes;
    }
    Close();
} //----- Fin du destructeur
```

```
/*************************************************************************
                    DotFileWriter – Écrivain de DOT-file
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

// Interface de la classe <DotFileWriter> (fichier DotFileWriter.h)
#if ! defined ( DOT_FILE_WRITER_H )
#define DOT_FILE_WRITER_H

//------------------------------------------------------------ Interfaces utilisées
#include <forward_list>
#include <fstream>
#include <string>

//------------------------------------------------------------------------ Types
struct Link
{
    unsigned long sourceNodeId;  // L'ID du noeud de source du lien.
    unsigned long destinationNodeId;  // L'ID du noeud de destination du lien.
    std::string label;  // L'étiquette à afficher à côté du lien.
};

//------------------------------------------------------------------------------
// Rôle de la classe <DotFileWriter>
// Transcrit un graphe au format Graphviz et permet de l'écrire dans un fichier
// DOT.
//------------------------------------------------------------------------------

class DotFileWriter
{
//------------------------------------------------------------------ PUBLIC
public:
//----------------------------------------------------------- Méthodes publiques
    void AddLink (
            unsigned long sourceId, unsigned long targetId,
            const std::string & linkLabel
    );
    // <sourceId> : l'ID du noeud de source du lien
    // <targetId> : l'ID du noeud de destination du lien
    // <linkLabel> : l'étiquette à afficher à côté du lien
    // Mode d'emploi : Ajoute un lien entre le noeud <sourceId> et le noeud
    // <targetId>, avec une étiquette <linkLabel> à côté.
    // Contrat : <sourceId> et <targetId> devraient déjà avoir été ajoutés grâce
    // à <AddNode>. Le graphe doit être initialisé avec InitGraph au préalable.

    void AddNode (unsigned long id, const std::string & label);
    // <id> : l'ID identifiant le noeud
    // <label> : l'étiquette à afficher à côté du noeud
    // Mode d'emploi : Ajoute un nouveau noeud du graphe, identifié par son
    // <id>. Son <label> est affiché à côté du noeud dans le graphe généré.
    // Contrat : Le graphe doit être initialisé avec InitGraph au préalable.

    void Close ();
    // Mode d'emploi : Ferme le flux de fichier associé, s'il a été ouvert. Si
    // aucun flux n'a été ouvert, aucune action n'est effectuée.

    void InitGraph(unsigned long graphNodes);
    // <graphNodes> : le nombre de noeuds dans le graphe
```

```
    // Mode d'emploi : Initialise le graphe associé, allouant assez de mémoire
    // pour stocker un nombre <graphNodes> de noeuds. Doit être appelé avant de
    // commencer à construire les noeuds et les liens. Si un graphe avait déjà
    // été initialisé, l'ancien graphe est supprimé et un nouveau est créé.

    bool Open (const std::string & filename);
    // <filename> : le nom de fichier à ouvrir en mode écriture
    // Mode d'emploi : Ouvre un fichier <filename> en mode écriture. Le fichier
    // devrait avoir une extension ".dot", mais ceci n'est pas obligatoire.

    void Write ();
    // Mode d'emploi : Écrit le fichier DOT associé au graphe initialisé par
    // InitGraph.
    // Contrat : Le graphe doit être initialisé avec InitGraph au préalable.

//------------------------------------------------------------ Surcharge d'opérateurs
    DotFileWriter & operator = (const DotFileWriter & writer) = delete;
    // <writer> : l'écrivain à copier
    // Mode d'emploi : Supprimé. La copie est interdite pour éviter d'avoir des
    // conflits liés aux flux.

//-------------------------------------------------- Constructeurs - destructeur
    DotFileWriter ();
    // Mode d'emploi : Initialise un écrivain de fichier DOT sans graphe
    // associé. Le graphe doit ensuite être initialisé avec InitGraph.

    DotFileWriter (const DotFileWriter & writer) = delete;
    // <writer> : l'écrivain à copier
    // Mode d'emploi : Supprimé. La copie est interdite pour éviter d'avoir des
    // conflits liés aux flux.

    virtual ~DotFileWriter ();
    // Mode d'emploi : Supprime l'écrivain, ferme tout flux de fichier associé
    // et supprime le graphe s'il a été initialisé.

//------------------------------------------------------------------------ PRIVE
protected:
//------------------------------------------------------ Attributs protégés
    std::ofstream dotFile;  // Le flux de fichier de sortie associé
    std::forward_list<Link> links;  // La liste de liens de graphe
    std::string * nodes;  // Le tableau de noeuds de graphe
    unsigned long numNodes;  // Le nombre de noeuds de graphe

    static const std::string HEADER;  // L'en-tête du fichier DOT à écrire
    static const std::string FOOTER;  // Le pied de page du fichier DOT à écrire
};

#endif // DOT_FILE_WRITER_H
```

```
/*************************************************************************
            HistoryManager - Gère un graphe de parcours d'un serveur
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <functional>
#include <sstream>

//------------------------------------------------------ Include personnel
#include "Document.h"
#include "DotFileWriter.h"
#include "HistoryManager.h"
#include "LogEntry.h"
#include "Logger.h"
#include "LogReader.h"

//------------------------------------------------------------- Constantes
const std::unordered_set<std::string> VALID_REQUEST_METHODS = {"GET", "POST"};
const unsigned short VALID_STATUS_CODES[2] = {200, 399};

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

//----------------------------------------------------- Méthodes publiques
bool HistoryManager::FromFile (
        LogReader & logFile,
        const std::unordered_set<std::string> & excludedExtensions,
        unsigned int startHour,
        unsigned int endHour
)
// Algorithme : Lit le fichier de log <logFile> ligne par ligne pour extraire
// les informations sur les documents parcourus, puis ne conserve que les
// documents satisfaisant les critères de filtre.
{
    DEBUG("Appel à HistoryManager::FromFile");
    while (!logFile.Eof())
    {
        LogEntry entry;
        try
        {
            logFile.ReadLine(entry);
        }
        catch (std::runtime_error & e)
        {
            WARNING(e.what());
            continue;
        }
        if (entry.GetHour() >= startHour && entry.GetHour() < endHour &&
            !excludedExtensions.count(entry.GetRequestUriExtension()) &&
            VALID_REQUEST_METHODS.count(entry.GetRequestMethod()) &&
            entry.GetStatusCode() >= VALID_STATUS_CODES[0] &&
            entry.GetStatusCode() <= VALID_STATUS_CODES[1])
        {
            addEntry(entry);
        }
```

```
    }
    return true;
} //----- Fin de FromFile

void HistoryManager::ListDocuments (unsigned int max) const
// Algorithme : Crée une copie triée (selon leur popularité, par ordre
// décroissant) de la liste des documents, puis affiche au plus <max> éléments
// de cette liste triée.
{
    DEBUG("Appel à HistoryManager::ListDocuments");
    Documents sortedDocuments = documents;
    std::sort(
            sortedDocuments.begin(), sortedDocuments.end(),
            std::greater<Document>()
    );
    for (Documents::size_type i=0, e=sortedDocuments.size(); i!=e && i<max; ++i)
    {
        if (sortedDocuments[i].GetLocalHits() == 0)
        {
            break;
        }
        std::cout << sortedDocuments[i].GetUri() << " (" <<
                sortedDocuments[i].GetLocalHits() << " hits)\n";
    }
    std::cout << std::flush;
} //----- Fin de ListDocuments

void HistoryManager::ToDotFile (DotFileWriter & dotFile) const
// Algorithme : Pour chaque document dans la liste de document, ajoute un noeud
// au DOT file, puis y ajoute tous les accès de ce document vers les autres
// documents comme des liens.
{
    DEBUG("Appel à HistoryManager::ToDotFile");
    dotFile.InitGraph(documents.size());
    for (Documents::size_type i = 0, e = documents.size(); i < e; ++i)
    {
        dotFile.AddNode(i, documents[i].GetUri());
        for (auto const & hit : documents[i].GetRemoteHits())
        {
            std::ostringstream ss;
            ss << hit.second;
            dotFile.AddLink(i, hit.first, ss.str());
        }
    }
    dotFile.Write();
} //----- Fin de ToDotFile

//-------------------------------------------------- Constructeurs - destructeur
HistoryManager::HistoryManager(const std::string & serverUrl) :
        localServerUrl(serverUrl)
{
    DEBUG("Appel au constructeur de HistoryManager");
} //----- Fin du constructeur

HistoryManager::~HistoryManager()
{
    DEBUG("Appel au destructeur de HistoryManager");
} //----- Fin du destructeur

//------------------------------------------------------------------------ PRIVE
```

```
//---------------------------------------------------------- Méthodes protégées
void HistoryManager::addEntry (const LogEntry & entry)
// Algorithme : Essaie de trouver le document demandé dans la liste de documents
// (en le créant s'il n'est pas trouvé), puis incrémente son compteur d'accès
// locaux de 1. Ensuite, l'algorithme trouve le document référant (en le créant
// s'il n'est pas trouvé), puis incrémente le nombre d'accès de ce document vers
// le document demandé.
{
    DEBUG("Appel à HistoryManager::addEntry");
    // Incrémenter le nombre d'accès au document demandé.
    std::string requestUri = entry.GetRequestUriConverted();
    Documents::size_type requestIndex;
    auto it = documentsByName.find(requestUri);
    if (it == documentsByName.end())
    {
        requestIndex = documents.size();
        documents.emplace_back(requestUri);
        documentsByName[documents[requestIndex].GetUri()] = requestIndex;
    }
    else
    {
        requestIndex = it->second;
    }
    documents[requestIndex].AddLocalHit();

    // Incrémenter le nombre d'accès à ce document au document référent.
    std::string refererUri = entry.GetRefererUrlConverted(localServerUrl);
    Documents::size_type refererIndex;
    it = documentsByName.find(refererUri);
    if (it == documentsByName.end())
    {
        refererIndex = documents.size();
        documents.emplace_back(refererUri);
        documentsByName[documents[refererIndex].GetUri()] = refererIndex;
    }
    else
    {
        refererIndex = it->second;
    }
    documents[refererIndex].AddRemoteHit(requestIndex);
} //----- Fin de addEntry
```

```
/**************************************************************************
           HistoryManager - Gère un graphe de parcours d'un serveur
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
**************************************************************************/

// Interface de la classe <HistoryManager> (fichier HistoryManager.h)
#if ! defined ( HISTORY_MANAGER_H )
#define HISTORY_MANAGER_H

//------------------------------------------------------------ Interfaces utilisées
#include <string>
#include <unordered_map>
#include <unordered_set>
#include <vector>

//------------------------------------------------------------------------------ Types
class Document;
class DotFileWriter;
class LogEntry;
class LogReader;

typedef std::vector<Document> Documents;
typedef std::unordered_map<std::string, Documents::size_type> NamedDocuments;

//------------------------------------------------------------------------------
// Rôle de la classe <HistoryManager>
// Gère un graphe d'historique de parcours d'un serveur Apache. Peut traiter les
// fichiers de log d'Apache pour en déduire les documents qui ont été parcourus.
//------------------------------------------------------------------------------

class HistoryManager
{
//------------------------------------------------------------------------------ PUBLIC
public:
//------------------------------------------------------------ Méthodes publiques
    bool FromFile (
            LogReader & logFile,
            const std::unordered_set<std::string> & excludedExtensions,
            unsigned int startHour,
            unsigned int endHour
    );
    // <logFile> : un lecteur de fichier Apache ouvert
    // <excludedExtensions> : la liste des extensions à exclure
    // <startHour> : l'heure de début pour la plage horaire (incluse)
    // <endHour> : l'heure de fin pour la plage horaire (exclue)
    // Mode d'emploi : Utilise le lecteur de logs Apache <logFile> pour extraire
    // l'historique de parcours d'un fichier. Seuls les documents avec des
    // extensions qui ne sont pas dans la liste <excludedExtensions> et qui
    // ont été visités dans la plage horaire [startHour; endHour[ sont traités.
    // Contrat : <logFile> doit être ouvert et disponible pour la lecture.

    void ListDocuments (unsigned int max) const;
    // <max> : le nombre maximum de documents à lister
    // Mode d'emploi : Affiche la liste des <max> documents les plus populaires
    // (ceux qui ont été le plus visité). S'il y a moins de documents que max,
    // tous les documents ayant été visités au moins une fois sont listés.

    void ToDotFile (DotFileWriter & dotFile) const;
    // <dotFile> : un écrivain de fichier DOT ouvert
```

```
    // Mode d'emploi : Génère un fichier DOT à partir de l'historique stocké. Le
    // fichier généré peut ensuite être passé à Graphviz pour génération d'une
    // image du graphe.
    // Contrat : <dotFile> doit être ouvert et disponible pour l'écriture.

//------------------------------------------------------ Constructeurs - destructeur
    HistoryManager (const std::string & serverUrl);
    // <serverUrl> : l'URL du serveur local associé à cet historique
    // Mode d'emploi : Construit un nouveau gestionnaire d'historique pour le
    // serveur spécifié par <serverUrl>. Si une URL commence par <serverUrl>,
    // le document associé est considéré comme faisant partie du serveur local.

    virtual ~HistoryManager ();
    // Mode d'emploi : Détruit le gestionnaire d'historique.

//------------------------------------------------------------------------------ PRIVE
protected:
//----------------------------------------------------------------- Méthodes protégées
    void addEntry (const LogEntry & entry);
    // <entry> : L'entrée à ajouter
    // Mode d'emploi : Ajoute une entrée d'un fichier de log à l'historique.
    // Chaque entrée correspond à un ou deux documents, qui sont ajoutés à la
    // liste de documents s'ils n'existent pas déjà ; sinon, leurs compteurs
    // respectifs sont incrémentés.

protected:
//----------------------------------------------------------------- Attributs protégés
    Documents documents;  // Le tableau de documents ; leur position correspond
                          // à leur ID
    NamedDocuments documentsByName;  // La liste associant l'URI des documents
                                     // à leur ID, pour faciliter leur recherche
                                     // par URI
    const std::string localServerUrl;  // L'URL du serveur local associé au
                                       // gestionnaire d'historique
};

#endif // HISTORY_MANAGER_H
```

```
/*************************************************************************
                LogEntry - Représente une ligne du fichier de log
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <cstdlib>
#include <stdexcept>

//------------------------------------------------------ Include personnel
#include "LogEntry.h"
#include "Logger.h"

//------------------------------------------------------------- Constantes
const std::string LogEntry::EXTERNAL_DOCUMENT = "*";
const regex LogEntry::REQUEST_URI(
        R"(^(?:\:\d+)?((?:\/(?:[^\/?]+\/)*)+([^.\/?][^\/?]+?)?(?:\.(\w*))?))"
        R"((?:((\?.*)|(;.+)))?$)"
);
const regex LogEntry::APACHE_LOG_ENTRY(
        R"(^(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3}) (\S+) (\S+) )"
        R"(\[(\d{2})\/(\w{3})\/(\d{4})\:(\d{2})\:(\d{2})\:(\d{2}) )"
        R"(\+(\d{2})(\d{2})] \"([A-Z]+) ([^\"]+?) +HTTP\/(\d\.\d)\" (\d+) )"
        R"((\d+|\-) \"([^\"]*)\" \"([^\"]*)\"$)"
);

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

//----------------------------------------------------- Fonctions amies
std::istream & operator >> (std::istream & input, LogEntry & logEntry)
// Algorithme : Passe la ligne à travers un regex pour en extraire les
// informations sous forme de string, puis convertit les résultats vers une
// forme appropriée.
{
    DEBUG("Appel à LogEntry::operator >>");
    std::string line;
    std::getline(input, line);
    if (line.empty())
    {
        return input;
    }
    smatch match;
    if (!regex_match(line, match, LogEntry::APACHE_LOG_ENTRY))
    {
        throw std::runtime_error("Invalid log entry");
    }
    logEntry.hour = atoi(match[10].str().c_str());
    logEntry.requestMethod = match[15];
    logEntry.requestUri = match[16];
    logEntry.statusCode = atoi(match[18].str().c_str());
    logEntry.refererUrl = match[20];
    return input;
} //----- Fin de operator >>

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

```
unsigned short LogEntry::GetHour () const
{
    DEBUG("Appel à LogEntry::GetHour");
    return hour;
} //----- Fin de GetHour

const std::string & LogEntry::GetRequestMethod () const
{
    DEBUG("Appel à LogEntry::GetRequestMethod");
    return requestMethod;
} //----- Fin de GetRequestMethod

const std::string LogEntry::GetRequestUriConverted () const
// Algorithme : Essaie de voir si l'URI a une forme spéciale (numéro de port,
// paramètres...) et ne renvoie que la partie de base si c'est le cas.
{
    DEBUG("Appel à LogEntry::GetRequestUriConverted");
    smatch match;
    if (!parseUri(requestUri, match))
    {
        return requestUri;
    }
    return match[1];
} //----- Fin de GetRequestUriConverted

const std::string LogEntry::GetRequestUriExtension () const
// Algorithme : Essaie d'interpréter l'URI avec un regex pour en extraire son
// extension de fichier.
{
    DEBUG("Appel à LogEntry::GetRequestUriExtension");
    smatch match;
    if (!parseUri(requestUri, match))
    {
        return "";
    }
    return match[3];
} //----- Fin de GetRequestUriExtension

unsigned short LogEntry::GetStatusCode () const
{
    DEBUG("Appel à LogEntry::GetStatusCode");
    return statusCode;
} //----- Fin de GetStatusCode

const std::string LogEntry::GetRefererUrlConverted (const std::string & local)
    const
// Algorithme : Essaie de simplifier l'URL du référant, en enlevant les
// paramètres, les numéros de port éventuels, et, pour les URL du serveur local,
// renvoie seulement l'URI du document, simplifiée.
{
    DEBUG("Appel à LogEntry::GetRefererUrlConverted");
    if (!refererUrl.compare(0, local.size(), local))
    {
        std::string convertedUrl = refererUrl.substr(local.size());
        if (convertedUrl.length() == 0)
        {
            return "/";  // insa-lyon.fr est équivalent à insa-lyon.fr/, donc on
                         // stocke le document avec l'URI / pour éviter les
                         // conflits.
        }
        smatch match;
```

```
        if (!parseUri(convertedUrl, match))
        {
            return convertedUrl;
        }
        return match[1];
    }
    return EXTERNAL_DOCUMENT;
} //----- Fin de GetRefererUrlConverted

//------------------------------------------------- Constructeurs - destructeur
LogEntry::LogEntry () :
        hour(0), requestMethod(""), requestUri(""), statusCode(0),
        refererUrl("")
{
    DEBUG("Appel au constructeur de LogEntry");
} //----- Fin du constructeur

LogEntry::~LogEntry ()
{
    DEBUG("Appel au destructeur de LogEntry");
} //----- Fin du destructeur

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

//------------------------------------------------------- Méthodes protégées
bool LogEntry::parseUri (const std::string & uri, smatch & match) const
{
    DEBUG("Appel à LogEntry::parseUri");
    return regex_match(uri, match, REQUEST_URI);
} //----- Fin de parseUri
```

```
/**************************************************************************
                LogEntry - Représente une ligne du fichier de log
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
**************************************************************************/

// Interface de la classe <LogEntry> (fichier LogEntry.h)
#if ! defined ( LOG_ENTRY_H )
#define LOG_ENTRY_H

//------------------------------------------------------------ Interfaces utilisées
#include <iostream>
#include <set>
#include <string>

// Utilisation de Boost si disponible, par souci de performance ; sinon, la
// bibliothèque regex de la STL est utilisée, par défaut.
#ifdef USE_BOOST
#include <boost/regex.hpp>
using boost::regex;
using boost::regex_match;
using boost::smatch;
#else
#include <regex>
using std::regex;
using std::regex_match;
using std::smatch;
#endif

//------------------------------------------------------------------------------
// Rôle de la classe <LogEntry>
// Représente une entrée dans un fichier de log Apache.
//------------------------------------------------------------------------------

class LogEntry
{
//---------------------------------------------------------------------- PUBLIC
public:
//------------------------------------------------------------ Fonctions amies
    friend std::istream & operator >> (
            std::istream & input, LogEntry & logEntry
    );
    // <input> : le flux d'entrée d'un fichier de log Apache
    // <logEntry> : l'entrée de log à remplir
    // Mode d'emploi : Permet de lire des entrées de log directement à partir
    // d'un flux, en extrayant les données importantes d'une ligne du fichier.

//------------------------------------------------------------ Méthodes publiques
    unsigned short GetHour () const;
    // Mode d'emploi : Renvoie l'heure de génération de l'entrée de log.

    const std::string & GetRequestMethod () const;
    // Mode d'emploi : Renvoie la méthode HTTP de requête du document.

    const std::string GetRequestUriConverted () const;
    // Mode d'emploi : Renvoie l'URI du document demandé par la requête HTTP.

    const std::string GetRequestUriExtension () const;
    // Mode d'emploi : Renvoie l'extension du document demandé. Si l'extension
    // ne peut être déduite, renvoie une chaîne vide.
```

```
    unsigned short GetStatusCode () const;
    // Mode d'emploi : Renvoie le statut de la requête.

    const std::string GetRefererUrlConverted (const std::string & local) const;
    // <local> : l'URL du serveur local
    // Mode d'emploi : Renvoie l'URL du référant de la requête. Si l'URL
    // commence par <local>, la requête vient du serveur local, et le préfixe
    // <local> est enlevé à l'URL.

//------------------------------------------------------ Constructeurs - destructeur
    LogEntry ();
    // Mode d'emploi : Construit une nouvelle entrée vide.

    virtual ~LogEntry ();
    // Mode d'emploi : Détruit l'entrée.

//------------------------------------------------------------------------------ PRIVE
protected:
//------------------------------------------------------------------ Méthodes protégées
    bool parseUri (const std::string & uri, smatch & match) const;
    // <uri> : l'URI à analyser
    // <match> : le résultat de l'examination, à modifier
    // Mode d'emploi : Examine une URI pour en extraire les informations
    // intéressantes.

protected:
//------------------------------------------------------------------ Attributs protégés
    unsigned short hour;  // L'heure de génération de la requête
    std::string requestMethod;  // La méthode de requête HTTP
    std::string requestUri;  // L'URI du document demandé
    unsigned short statusCode;  // Le statut de la requête
    std::string refererUrl;  // L'URL du référant de la requête

    static const std::string EXTERNAL_DOCUMENT;  // L'étiquette spéciale pour
                                                 // les documents externes
    static const regex APACHE_LOG_ENTRY;  // Le regex pour une ligne du log
    static const regex REQUEST_URI;  // Le regex pour traiter une URI
};

#endif // LOG_ENTRY_H
```

```
/*************************************************************************
                Logger - Gère l'affichage de messages de log
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include personnel
#include "Logger.h"

//------------------------------------------------------------------ Constantes
const std::string Logger::P = "\033[";

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

//-------------------------------------------------------- Méthodes publiques
void Logger::print (std::ostream & out)
{
    out << std::endl;
}
```

```
/*************************************************************************
                Logger - Gère l'affichage de messages de log
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

// Interface de la classe <Logger> (fichier Logger.h)
#if ! defined ( LOGGER_H )
#define LOGGER_H

// Définir les macros de niveau de log.
#ifdef MAP
#define DEBUG(args ...) Logger::Debug(args)
#else
#define DEBUG(args ...)
#endif // MAP
#define ERROR(args ...) Logger::Error(args)
#define INFO(args ...) Logger::Info(args)
#define WARNING(args ...) Logger::Warning(args)

//------------------------------------------------------------ Interfaces utilisées
#include <iostream>
#include <string>

//------------------------------------------------------------------------------ Types
enum TerminalColor
{
    NONE = 0,
    BLACK,
    RED,
    GREEN,
    YELLOW,
    BLUE,
    MAGENTA,
    CYAN,
    WHITE
};

//------------------------------------------------------------------------------------
// Rôle de la classe <Logger>
// Classe d'utilité pour afficher des messages sur la sortie standard. Permet de
// choisir des couleurs pour les messages.
//------------------------------------------------------------------------------------

class Logger
{
//------------------------------------------------------------------------------ PUBLIC
public:
//------------------------------------------------------------ Méthodes publiques

    template<typename ... ARGS> static void Debug (ARGS ... args);
    // <args> : un nombre variable d'éléments à afficher
    // Mode d'emploi : Affiche un message de niveau debug si la définition MAP
    // est active. Le préfixe "Debug: " est ajouté au message. Si les couleurs
    // sont actives, le message sera affiché en vert.

    template<typename ... ARGS> static void Error (ARGS ... args);
    // <args> : un nombre variable d'éléments à afficher
    // Mode d'emploi : Affiche un message de niveau erreur. Le préfixe "Error :"
    // est ajouté au message. Si les couleurs sont actives, le message sera
```

```
    // affiché en rouge.

    template<typename ... ARGS> static void Info (ARGS ... args);
    // <args> : un nombre variable d'éléments à afficher
    // Mode d'emploi : Affiche un message de niveau information. Le message n'a
    // aucune couleur spéciale.

    template<typename ... ARGS> static void Warning (ARGS ... args);
    // <args> : un nombre variable d'éléments à afficher
    // Mode d'emploi : Affiche un message de niveau avertissement. Le préfixe
    // "Warning: " est ajouté au message. Si les couleurs sont actives, le
    // message sera affiché en jaune.

//------------------------------------------------------------------------- PRIVE
protected:
//----------------------------------------------------------------- Méthodes protégées
    template<typename ... ARGS> static void log (
            std::ostream & out, TerminalColor color,
            ARGS ... args
    );
    // <out> : le flux de sortie des messages
    // <color> : la couleur du message
    // <args> : un nombre variable d'éléments à afficher
    // Mode d'emploi : Affiche un message sur le flux de sortie <out>, avec la
    // couleur <color>. Le message est constitué de la somme de tous les <args>.

    static void print (std::ostream & out);
    // <out> : le flux de sortie des messages
    // Mode d'emploi : Appelé en fin de message, effectue un retour à la ligne.

    template<typename FIRST, typename ... ARGS> static void print (
            std::ostream & out, FIRST arg1, ARGS ... args
    );
    // <out> : le flux de sortie des messages
    // <args1> : l'argument à envoyer dans le flux <out>
    // <args> : le reste des arguments à afficher par la suite
    // Mode d'emploi : S'appelle récursivement avec une liste variable
    // d'arguments. Quand il ne reste plus d'arguments, un retour à la ligne est
    // effectué.

protected:
//------------------------------------------------------------------- Attributs protégés
    static const std::string P;  // Le préfixe des codes couleur
};

//-------------------------------------------------------- Implémentations de <Logger>
template<typename ... ARGS> void Logger::Debug (ARGS ... args)
{
    log(std::cout, GREEN, "Debug: ", args ...);
} //----- Fin de Debug

template<typename ... ARGS> void Logger::Error (ARGS ... args)
{
    log(std::cerr, RED, "Error: ", args ...);
} //----- Fin de Error

template<typename ... ARGS> void Logger::Info (ARGS ... args)
{
    log(std::cout, NONE, args ...);
} //----- Fin de Info
```

```
template<typename ... ARGS> void Logger::Warning (ARGS ... args)
{
    log(std::cerr, YELLOW, "Warning: ", args ...);
} //----- Fin de Warning

template<typename ... ARGS>
void Logger::log (std::ostream & out, TerminalColor color, ARGS ... args)
// Algorithme : Insère les codes couleur si COLORS est actif ; sinon, affiche le
// message sans couleur.
{
#ifdef COLORS
    if (color == NONE)
    {
#endif // COLORS
        print(out, args ...);
#ifdef COLORS
    }
    else
    {
        print(out, P, 29 + color, "m", args ..., P, "0m");
    }
#endif // COLORS
} //----- Fin de log

template<typename FIRST, typename ... ARGS>
void Logger::print (std::ostream & out, FIRST arg1, ARGS ... args)
// Algorithme : Affiche le premier argument, puis s'appelle récursivement
// jusqu'à ce qu'il ne reste plus d'arguments.
{
    out << arg1;
    print(out, args ...);
}; //----- Fin de print

#endif // LOGGER_H
```

```
/*************************************************************************
                LogReader - Lecteur de fichier de log Apache
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

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

//-------------------------------------------------------- Include système
#include <stdexcept>
#include <sstream>

//------------------------------------------------------ Include personnel
#include "LogEntry.h"
#include "Logger.h"
#include "LogReader.h"

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

//----------------------------------------------------- Méthodes publiques
void LogReader::Close ()
{
    DEBUG("Appel à LogReader::Close");
    if (logFile.is_open())
    {
        logFile.close();
    }
} //----- Fin de Close

bool LogReader::Eof () const
{
    DEBUG("Appel à LogReader::Eof");
    return logFile.eof();
} //----- Fin de Eof

bool LogReader::Open (const std::string & filename)
// Algorithme : Ferme tout flux déjà ouvert et ouvre un nouveau lié au fichier
// <filename>.
{
    DEBUG("Appel à LogReader::Open");
    Close();
    currentLine = 0;
    logFile.open(filename);
    return logFile.is_open();
} //----- Fin de Open

void LogReader::ReadLine (LogEntry & entry)
// Algorithme : Lit une seule ligne du fichier, si encore possible, puis en
// extrait les informations intéressantes pour les stocker dans <entry>
{
    DEBUG("Appel à LogReader::ReadLine");
    currentLine++;
    try
    {
        logFile >> entry;
    }
    catch (std::runtime_error & e)
    {
```

```
        std::stringstream ss;
        ss << "Failed to parse line " << currentLine << " (" << e.what() << ")";
        throw std::runtime_error(ss.str());
    }
} //----- Fin de ReadLine

//------------------------------------------------------ Constructeurs - destructeur
LogReader::LogReader () : currentLine(0)
{
    DEBUG("Appel au constructeur de LogReader");
} //----- Fin du constructeur

LogReader::~LogReader ()
{
    DEBUG("Appel au destructeur de LogReader");
    Close();
} //----- Fin du destructeur
```

```
/*************************************************************************
                LogReader - Lecteur de fichier de log Apache
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

// Interface de la classe <LogReader> (fichier LogReader.h)
#if ! defined ( LOG_READER_H )
#define LOG_READER_H

//------------------------------------------------------------ Interfaces utilisées
#include <fstream>
#include <string>

//------------------------------------------------------------------------ Types
class LogEntry;

//------------------------------------------------------------------------------
// Rôle de la classe <LogReader>
// Permet de lire un fichier de log Apache pour en extraire les données sur
// chaque ligne.
//------------------------------------------------------------------------------

class LogReader
{
//------------------------------------------------------------------------ PUBLIC
public:
//----------------------------------------------------------- Méthodes publiques
    void Close ();
    // Mode d'emploi : Ferme le flux de fichier associé, s'il a été ouvert. Si
    // aucun flux n'a été ouvert, aucune action n'est effectuée.

    bool Eof () const;
    // Mode d'emploi : Renvoie vrai si la fin de fichier a été atteinte.

    bool Open (const std::string & filename);
    // <filename> : le nom de fichier à ouvrir en mode lecture
    // Mode d'emploi : Ouvre un fichier de log <filename> en mode lecture.

    void ReadLine (LogEntry & entry);
    // <entry> : l'entrée à remplir d'informations
    // Mode d'emploi : Lit une ligne du fichier et remplit <entry> des
    // informations extraites de la ligne.

//------------------------------------------------- Surcharge d'opérateurs
    LogReader & operator = (const LogReader & reader) = delete;
    // <reader> : le lecteur à copier
    // Mode d'emploi : Supprimé. La copie est interdite pour éviter d'avoir des
    // conflits liés aux flux.

//-------------------------------------------- Constructeurs - destructeur
    LogReader ();
    // Mode d'emploi : Initialise un lecteur sans flux ouvert.

    LogReader (const LogReader & reader) = delete;
    // <reader> : le lecteur à copier
    // Mode d'emploi : Supprimé. La copie est interdite pour éviter d'avoir des
    // conflits liés aux flux.

    virtual ~LogReader ();
```

```
    // Mode d'emploi : Supprime le lecteur et ferme tout flux de fichier
    // associé.

//-------------------------------------------------------------------------- PRIVE
protected:
//----------------------------------------------------------------- Attributs protégés
    std::ifstream logFile;  // Le flux de fichier d'entrée associé
    int currentLine;  // Le numéro de la ligne actuelle de lecture
};

#endif // LOG_READER_H
```

```
/*************************************************************************
                main - Interpréteur à la ligne de commande
                             -------------------
    début                : 01/12/2015
    copyright            : (C) 2015 par B3309
*************************************************************************/

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

//---------------------------------------------------------- Include système
using namespace std;
#include <list>
#include <string>
#include <tclap/CmdLine.h>
#include <unordered_set>

//---------------------------------------------------------- Include personnel
#include "ConfigReader.h"
#include "DotFileWriter.h"
#include "HistoryManager.h"
#include "Logger.h"
#include "LogReader.h"

//---------------------------------------------------------------- Constantes
// La description du programme, affichée à l'appel de --help
const string DESCRIPTION = "Parser for Apache logs";

// La version en cours du programme, affiché à l'appel de --version
const string VERSION = "1.0";

// Le nom du fichier de configuration optionnel
const string CONFIG_FILENAME = "tp-oo_3.cfg";

// La liste des extensions de fichier à exclure par défaut lorsque l'option -e
// est spécifiée
const unordered_set<string> DEFAULT_EXCLUDED_EXTENSIONS = {
        // Images
        "art", "bm", "bmp", "dwg", "dxf", "fig", "flo", "fpx", "g3", "gif",
        "ico", "ief", "iefs", "jfif", "jfif-tbnl", "jpe", "jpeg", "jpg", "jps",
        "jut", "mcf", "nap", "naplps", "nif", "niff", "pbm", "pct", "pcx",
        "pgm", "pic", "pict", "pm", "png", "pnm", "ppm", "qif", "qti", "qtif",
        "ras", "rast", "rf", "rgb", "rp", "svg", "svf", "tif", "tiff", "turbot",
        "wbmp", "xbm", "xif", "xpm", "xwd",
        // CSS
        "css",
        // JavaScript
        "js"
};

// La racine de l'URL du serveur de base par défaut
const string DEFAULT_LOCAL_URL = "http://intranet-if.insa-lyon.fr";

// Le nombre maximal de documents à afficher par défaut
const unsigned int DEFAULT_MAX_DOCUMENTS = 10;

//------------------------------------------------------------------ FONCTIONS
int main (int argc, const char * const * argv)
// Algorithme : Construit le lecteur d'arguments, traite l'entrée pour voir
// quels arguments ont été passés, puis effectue toutes les opérations
// demandées par l'utilisateur en paramètre.
{
```

```
ConfigReader config(CONFIG_FILENAME);
string dotFilename;
string logFilename;
DotFileWriter dotFile;
LogReader logFile;
unordered_set<string> excludedExtensions;
unsigned int startHour = 0, endHour = 24;

// Désactiver la synchronization avec la bibliothèque IO de C.
cout.sync_with_stdio(false);

// Initialiser le parseur d'arguments.
TCLAP::CmdLine cmd(DESCRIPTION, ' ', VERSION);

// Configurer l'argument du chemin de fichier log.
TCLAP::UnlabeledValueArg<string> logFilenameArg(
        "log", "path to the Apache log file to parse", true, "", "FILE",
        cmd
);

// Configurer l'argument (optionnel) du chemin de fichier dot-file, pour
// la génération d'un graphe Graphviz.
TCLAP::ValueArg<string> dotFilenameArg(
        "g", "graphviz", "path to a Graphviz file to generate", false, "",
        "FILE", cmd
);

TCLAP::SwitchArg excludeExtensionsArg(
        "e", "exclude", "exclude restricted extensions", cmd, false
);

// Configurer l'argument (optionnel) de restriction de la plage horaire.
// Seules des valeurs comprises entre 0 et 23 (incluses) sont acceptées.
vector<unsigned int> allowedHours;
for (unsigned int i = 0; i < 24; i++)
{
    allowedHours.push_back(i);
}
TCLAP::ValuesConstraint<unsigned int> timeVals( allowedHours );
TCLAP::ValueArg<unsigned int> timeArg(
        "t", "time", "restrict parsing to the specified hour", false,
        0, &timeVals, cmd
);

// Récupérer les valeurs des arguments à la ligne de commande.
try
{
    cmd.parse(argc, argv);

    // Essayer de lire le fichier de log.
    logFilename = logFilenameArg.getValue();
    if (!logFile.Open(logFilename))
    {
        ERROR("Failed to open log file for reading");
        return 1;
    }

    // Essayer de lire le DOT-file, si demandé.
    if (dotFilenameArg.isSet())
    {
        dotFilename = dotFilenameArg.getValue();
```

```
            if (!dotFile.Open(dotFilename))
            {
                ERROR("Failed to open DOT file for writing");
                return 1;
            }
        }

        // Enregistrer les extensions à exclure de la lecture du log.
        if (excludeExtensionsArg.getValue())
        {
            excludedExtensions = config.GetSet(
                    "EXCLUDED_EXTENSIONS", DEFAULT_EXCLUDED_EXTENSIONS
            );
        }

        // Restreindre la plage horaire autorisée, si demandée.
        if (timeArg.isSet())
        {
            startHour = timeArg.getValue();
            endHour = startHour + 1;
            WARNING(
                    "Only hits between ", startHour, "h and ", endHour,
                    "h have been taken into account."
            );
        }
    }
    catch (TCLAP::ArgException & e)
    {
        ERROR(e.what());
        return 1;
    }

    // Peupler l'historique de documents à partir du fichier log.
    HistoryManager historyMgr(config.GetString("LOCAL_URL", DEFAULT_LOCAL_URL));
    bool loaded = historyMgr.FromFile(
            logFile, excludedExtensions, startHour, endHour
    );
    logFile.Close();
    if (!loaded)
    {
        ERROR("Failed to parse log file");
        return 1;
    }

    // Générer le dot-file, si demandé.
    if (dotFilenameArg.isSet())
    {
        historyMgr.ToDotFile(dotFile);
        dotFile.Close();
        INFO("Dot-file ", dotFilename, " generated");
    }

    // Afficher les documents les plus populaires.
    historyMgr.ListDocuments(
            static_cast<unsigned int>(
                    config.GetInteger("MAX_DOCUMENTS", DEFAULT_MAX_DOCUMENTS)
            )
    );

    return 0;
} //----- Fin de main
```