TP3 C++ : CODE SOURCE

HAFEDH Jebalia

SOTIR Paul-Emmanuel Binôme B3311

# Main.cpp

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

main.cpp

--------

date                 : 12/2015

copyright            : (C) 2015 by B3311

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

//----------------------------------------------------------------------- INCLUDES

//-------------------------------------------------------------- Includes systèmes

#include <array>

#include <string>

#include <fstream>

#include <utility>

#include <iterator>

#include <iostream>

#include <algorithm>

//------------------------------------------------------------ Includes personnels

#include "Graph.h"

#include "Help\_txt.h"

#include "Log\_parser.h"

#include "Typed\_main\_binding.h"

namespace TP3

{

//------------------------------------------------------------ COMMAND OPTIONS

//! @remarks valide le concept de 'option\_with\_value'

struct input\_log\_file\_option

{ std::string value; };

//! @remarks valide le concept de 'option\_with\_tags' et 'option\_with\_value'

struct output\_graph\_file\_option

{

std::string value;

static tags\_t<3> get\_option\_tags() noexcept { return { "-g", "-G", "--graph"}; }

};

//! @remarks valide le concept de 'option\_with\_tags' et 'option\_with\_value'

struct hour\_option

{

unsigned short value;

static tags\_t<3> get\_option\_tags() noexcept { return { "-t", "-T", "--hour" }; }

};

//! @remarks valide le concept de 'option\_with\_tags' et 'option\_with\_value'

struct list\_option

{

size\_t value;

static const size\_t default\_value = 10;

static tags\_t<3> get\_option\_tags() noexcept { return { "-l", "-L", "--listCount" }; }

};

//! @remarks valide le concept de 'option\_with\_tags'

struct exclusion\_option

{ static tags\_t<3> get\_option\_tags() noexcept { return { "-e", "-E", "--excludeMedias" }; } };

//! @remarks valide le concept de 'option\_with\_tags'

struct help\_option

{ static tags\_t<1> get\_option\_tags() noexcept { return { "-h" }; } };

using std::experimental::optional;

//----------------------------------------------------------- STATIC FUNCTIONS

//! "Main typé" appellé par un objet de type 'Typed\_main\_binding<Args...>'

static void typed\_main(optional<input\_log\_file\_option> input\_log,

optional<output\_graph\_file\_option> output\_graph,

optional<hour\_option> hour\_opt,

optional<list\_option> list\_count\_opt,

optional<exclusion\_option> excl\_opt,

optional<help\_option> help)

{

if (!help)

{

Log\_parser parser;

if (hour\_opt)

{

if (hour\_opt->value < 0 || hour\_opt->value >= 24)

throw std::out\_of\_range("Hour must be in [[0,23]]");

parser.enable\_hour\_filter(hour\_opt->value);

}

if (excl\_opt)

parser.enable\_exclusion();

if (output\_graph)

{

// Parse and serialize graph

serialize\_graph(output\_graph->value, parser.parse\_graph(input\_log->value));

std::cout << "Dot-file " << output\_graph->value << " generated." << std::endl;

}

else

{

// Parse log to obtain the URLs with their respective occurrence number (unordered\_multimap)

auto urls = parser.parse\_urllist(input\_log->value);

// Determine URLs toplist size

auto toplist\_count = list\_count\_opt ? list\_count\_opt->value : list\_option::default\_value;

if (toplist\_count < 1)

throw std::out\_of\_range("URLs toplist size must be greater than 0");

// Get top URL listS from unordered multimap '\*urls' (partial sort by URL occurrence number)

using url\_score\_t = std::pair<std::string, unsigned int>;

std::vector<url\_score\_t> topten(std::min(toplist\_count, urls->size()));

std::partial\_sort\_copy(std::begin(\*urls), std::end(\*urls), std::begin(topten), std::end(topten),

[](const url\_score\_t& a, const url\_score\_t& b)

{ return b.second < a.second; });

// Display the URLs top 10

for (const auto& score : topten)

std::cout << '"' << score.first << "\" :\t" << score.second << " occurrences" << std::endl;

}

}

else

std::cout << Help\_txt();

}

}

//--------------------------------------------------------------------------- MAIN

//! Fonction main utilisant simplement un binding vers le main typé

int main(int argc, char\* argv[])

{

// On traite le cas où seul '-h' est spécifié à part car le binding considère le nom du fichier log comme obligatoire

if (argc == 2)

if (argv[1] == std::string("-h"))

{

std::cout << TP3::Help\_txt();

return EXIT\_SUCCESS;

}

auto main\_binding = TP3::make\_typed\_main\_binding(&TP3::typed\_main);

try

{

main\_binding.exec(argc, argv);

}

catch (std::invalid\_argument e) { std::cout << "ERROR: " << e.what() << std::endl; }

catch (std::out\_of\_range e) { std::cout << "ERROR: " << e.what() << std::endl; }

return EXIT\_SUCCESS;

}

# Utils.h

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

Utils  -  A few common functions or types

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

date                 : 12/2015

copyright            : (C) 2015 by B3311

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

#ifndef UTILS\_H

#define UTILS\_H

#include <tuple>

#include <string>

namespace TP3

{

//----------------------------------------------- Variadic template indexing utils

//! Obtient le Nème type T issu du variadic template 'Args...'

template <size\_t N, typename... Args>

using get\_by\_Index\_t = typename std::tuple\_element<N, std::tuple<Args...>>::type;

//! Metafunction recursive obtenant l'indice du premier type T dans le variadic template 'Args...'

template <typename T, typename... Args>

struct index\_of;

//! Metafunction recursive obtenant l'indice du premier type T dans le variadic template 'Args...' (arret)

template <typename T, typename... Args>

struct index\_of<T, T, Args...> : std::integral\_constant<std::size\_t, 0> { };

//! Metafunction recursive obtenant l'indice du premier type T dans le variadic template 'Args...' (récursion)

template <typename T, typename Tail, typename... Args>

struct index\_of<T, Tail, Args...> : std::integral\_constant<std::size\_t, 1 + index\_of<T, Args...>::value> { };

//---------------------------------------------------------------- Positive modulo

template<typename T>

constexpr T positive\_mod(T a, T b)

{

return ((a % b) + b) % b;

}

//----------------------------------------------------------------- Parse<T, bool>

//! Fonction template convertissant une 'std::string' vers le type T

template<typename T, bool strong\_convertion = true>

inline T parse(const std::string& str)

{

return static\_cast<T>(str);

}

template<>

inline float parse<float, true>(const std::string& str)

{

size\_t idx;

auto val = std::stof(str, &idx);

if (idx != str.length())

throw std::invalid\_argument("Given std::string cannot be converted to float");

return val;

}

template<>

inline float parse<float, false>(const std::string& str) { return std::stof(str); }

template<>

inline double parse<double, true>(const std::string& str)

{

size\_t idx;

auto val = std::stod(str, &idx);

if (idx != str.length())

throw std::invalid\_argument("Given std::string cannot be converted to double");

return val;

}

template<>

inline double parse<double, false>(const std::string& str) { return std::stod(str); }

// ... (spécialisations de parse pour ‘long double’, ‘int’, ‘unsigned in’, ‘short’, ‘unisgned short’, ‘long’,

// ‘unsigned long’, ‘long long’ et ‘unsigned long long’ )

}

#endif // UTILS\_H

# Typed\_main\_binding.h

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

Typed\_main\_binding  - An helper class for main args

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

date                 : 12/2015

copyright            : (C) 2015 by B3311

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

//---------- Interface de la classe template Typed\_main\_binding<Args...> ---------

#ifndef TYPED\_MAIN\_BINDING\_H

#define TYPED\_MAIN\_BINDING\_H

//-------------------------------------------------------------- Includes systèmes

#include <vector>

#include <utility>

#include "optional.h"

#include <type\_traits>

//------------------------------------------------------------ Includes personnels

#include "Utils.h"

namespace TP3

{

//-------------------------------------------------------------- Metafonctions

//! Alias simplifiant la spécification du type de retour des méthodes 'get\_option\_tags()'

//! @remarks

//! Les seules contraintes concernant le concept de 'option\_with\_tags' est que le type

//! de retour de 'get\_option\_tags()' soit itérable et contienne des valeurs implicitement

//! convertibles en 'std::string' (pas nescessairement statique ou de type tags\_t<N>)

template<size\_t N>

using tags\_t = std::array<std::string, N>;

//! Spécialisation d'une metafonction aidant à verifier si un type T n'est pas conforme

//! au concept de 'option\_with\_tags'

template<typename T, typename = void>

struct has\_option\_tag\_getter : std::false\_type { };

//! Spécialisation d'une metafonction aidant à verifier si un type T est conforme au

//! concept de 'option\_with\_tags' (verifie seulement la présence de la méthode statique)

template<typename T>

struct has\_option\_tag\_getter<T, decltype(std::declval<T>().get\_option\_tags(), void())> : std::true\_type { };

//! Spécialisation d'une metafonction aidant à verifier si un type T n'est pas conforme

//! au concept de 'option\_with\_value'

template<typename T, typename = void>

struct option\_has\_value : std::false\_type { };

//! Spécialisation d'une metafonction aidant à verifier si un type T est conforme au

//! concept de 'option\_with\_value'

template<typename T>

struct option\_has\_value<T, decltype(std::declval<T>().value, void())> : std::true\_type { };

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

//! Classe template permettant de simplifier l'interprétation de argv et argc.

//! A partir d'une fonction dont la signature est du type 'typed\_main\_t<Args...>'

//! (avec les types Args validants les concepts décrits ci dessous), la classe

//! template en déduit statiquement la forme des commandes acceptables et gère

//! automatiquement toutes les vérifications et conversions nécessaires au runtime

//! pour pouvoir appeler, si possible, le main typé spécifié (avec les paramètres

//! optionnels correspondants à la commande).

//! @remarks

//! Les arguments templates 'Args' doivent être conformes aux concepts de :

//! - 'option\_with\_tags' et 'option\_with\_value' : option ayant un ou des tags

//! et un paramètre (e.g. "-t 21")

//! - ou 'option\_with\_tags' : option sans paramètre avec un tag (e.g. "-e")

//! - ou 'option\_with\_value' : option sans tags et ayant un paramètre.

//! Ce type d'option est en fait obligatoire : il determine la forme de la

//! commande car toute les options de (Args...) avant celle-ci doivent être

//! également avant dans argv (de même pour les options après une option sans tag)

//! @remarks

//! - Description du concept 'option\_with\_value' : Le type T doit contenir un attribut

//! publique 'value' tel que son type soit pourvus d'une surcharge de 'T TP3::parse<T>(std::string)'

//! - Description du concept 'option\_with\_tags' : Le type T doit contenir une méthode

//! publique et statique 'get\_option\_tags()' retournant un objet itérable contenant

//! des valeurs implicitment convertibles en 'std::string'.

// TODO: support default values

// TODO: allow multiple typed\_main implementation for alternative command usages

template<typename... Args>

class Typed\_main\_binding

{

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

public:

//----------------------------------------------- Types et alias publiques

using typed\_main\_t = std::function<void(std::experimental::optional<Args>...)>;

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

//! Execute, si possible, le main typé en construisant les paramètres optionels

//! du main typé à partir de argv et argc.

//! @throw std::invalid\_argument

//! @throw std::out\_of\_range

void exec(int argc, char\* argv[]);

//----------------------------------------------------- Méthodes spéciales

//! Constructeur prenant en paramètre le main typé qui sera executé

explicit Typed\_main\_binding(const typed\_main\_t& typed\_main);

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

private:

//------------------------------------------------------- Méthodes privées

//! spécialisation template utilisant SFINAE pour cibler les options ayant

//! seulement un tag (validant le concept de 'option\_with\_tags')

template<typename T,

typename = typename std::enable\_if<!option\_has\_value<T>::value && has\_option\_tag\_getter<T>::value>::type>

std::experimental::optional<T> get\_opt(int argc, char\* argv[]);

//! spécialisation template utilisant SFINAE pour cibler les options ayant

//! un parametre (validant les concepts de 'option\_with\_tags' et 'option\_with\_value')

template<typename T,

typename = typename std::enable\_if<option\_has\_value<T>::value && has\_option\_tag\_getter<T>::value>::type,

typename = void>

std::experimental::optional<T> get\_opt(int argc, char\* argv[]);

//! spécialisation template utilisant SFINAE pour cibler les options ne

//! disposant pas de tags (validant le concept de 'option\_with\_value')

template<typename T,

  typename = typename std::enable\_if<option\_has\_value<T>::value && !has\_option\_tag\_getter<T>::value>::type,

  typename = void, typename = void>

std::experimental::optional<T> get\_opt(int argc, char\* argv[]);

// TODO: ajouter une spécialisation de get\_opt pour gérer les paramètres multiples pour une même option

//------------------------------------------------------- Attributs privés

typed\_main\_t m\_typed\_main\_func;

std::vector<std::pair<size\_t, int>> m\_found\_parameters;

//---------------------------------------------------- Fonctions statiques

//! Fonction permettant la construction d'un tableau des tags de tout les types Args...

template<typename T, typename = typename std::enable\_if<has\_option\_tag\_getter<T>::value>::type>

static std::vector<std::string> get\_option\_tags();

//! Fonction permettant la construction d'un tableau des tags de tout les types Args...

template<typename T, typename = typename std::enable\_if<!has\_option\_tag\_getter<T>::value>::type, typename = void>

static std::vector<std::string> get\_option\_tags();

};

//--------------------------------------------------------- Fonctions globales

//! Fonction créant un objet de type 'Typed\_main\_binding<Args...>' à partir d'un pointeur de fonction vers un

//! main typé.

//! @remarks This function permits implicit template deduction by taking C-style function pointer instead of

//! std::function<...>.

template<typename... Args>

Typed\_main\_binding<Args...> make\_typed\_main\_binding(void(\*typed\_main)(std::experimental::optional<Args>...) )

{

return Typed\_main\_binding<Args...>(std::function<void(std::experimental::optional<Args>...)> (typed\_main));

}

//----- Implémentation de la classe template Typed\_main\_binding<Args...> -----

template<typename... Args>

Typed\_main\_binding<Args...>::Typed\_main\_binding(const typed\_main\_t& typed\_main)

: m\_typed\_main\_func(typed\_main) { }

template<typename... Args>

void Typed\_main\_binding<Args...>::exec(int argc, char\* argv[])

{

if (argc > 0 && argv != nullptr)

{

int arg\_processed\_cout = 1;

// Store an array of booleans indicating Args...'s option type (tag presence and value presence)

std::array<std::pair<bool, bool>, sizeof...(Args)> options\_types\_info =

decltype(options\_types\_info){ { std::make\_pair(has\_option\_tag\_getter<Args>::value,

option\_has\_value<Args>::value)... }};

// Store an array of options tags

std::array<std::vector<std::string>, sizeof...(Args)> options\_tags =

decltype(options\_tags){ { get\_option\_tags<Args>()... }};

// Find presence of options with tags in argv

int previous\_found\_option\_param\_count = 0;

int previous\_found\_option\_idx = 0;

std::array<bool, sizeof...(Args)> processed\_template\_arg\_indices;

processed\_template\_arg\_indices.fill(false);

for (auto arg\_idx = 1; arg\_idx < argc; ++arg\_idx)

{

std::string cmd\_str(argv[arg\_idx]);

bool found = false;

for (size\_t T\_idx = 0; T\_idx < sizeof...(Args) && !found; ++T\_idx)

{

auto type\_info = options\_types\_info[T\_idx];

if(type\_info.first) // if T has option tag getter (ignore mandatory/tagless parameters for now)

{

auto T\_tags = options\_tags[T\_idx];

if (T\_tags.size() == 0)

throw std::invalid\_argument("Option with tags don't have any tag");

for (const auto& tag : T\_tags)

{

if (tag == cmd\_str)

{

if(arg\_idx <= previous\_found\_option\_param\_count + previous\_found\_option\_idx)

throw std::invalid\_argument("No parameter after an option tag which need one");

previous\_found\_option\_param\_count = type\_info.second ? 1 : 0;

arg\_processed\_cout += 1 + previous\_found\_option\_param\_count;

if (processed\_template\_arg\_indices[T\_idx])

throw std::invalid\_argument("Multiple tag for the same option");

m\_found\_parameters.emplace\_back(T\_idx, arg\_idx);

processed\_template\_arg\_indices[T\_idx] = true;

previous\_found\_option\_idx = arg\_idx;

found = true;

break;

}

}

}

}

}

// Check mandatory/tagless parameters presence

std::vector<std::pair<size\_t, size\_t>> found\_tagless\_options;

for (size\_t T\_idx = 0; T\_idx < sizeof...(Args); ++T\_idx)

{

auto type\_info = options\_types\_info[T\_idx];

if (!type\_info.first) // if T has option tag getter

{

int idx = 1;

bool reached\_later\_option = false;

for(const auto& p : m\_found\_parameters)

{

if (reached\_later\_option && p.second == idx)

// mandatory/tagless parameter index is actually used by an option with a tag

throw std::invalid\_argument("Missing mandatory tagless option");

if(p.first < T\_idx)

{

if (reached\_later\_option)

throw std::invalid\_argument("Invalid command shape (options order imposed

by tagless option(s) not respected)");

idx = p.second + type\_info.second ? 2 : 1; // if T has option parameter/value

}

else if(p.first > T\_idx && !reached\_later\_option)

{

if (p.second == idx)

throw std::invalid\_argument("Missing mandatory tagless option or invalid command shape");

reached\_later\_option = true;

found\_tagless\_options.emplace\_back(T\_idx, idx);

arg\_processed\_cout++;

}

}

if(!reached\_later\_option && idx < argc)

{

found\_tagless\_options.emplace\_back(T\_idx, idx);

arg\_processed\_cout++;

}

}

}

if(arg\_processed\_cout < argc)

throw std::invalid\_argument("Invalid options");

m\_found\_parameters.insert(std::end(m\_found\_parameters),

std::begin(found\_tagless\_options),

std::end(found\_tagless\_options));

// We have verified command shape and found all options/parameters of the command, we can now parse parameters

// and call typed main:

m\_typed\_main\_func((get\_opt<Args>(argc, argv))...);

}

}

template<typename... Args>

template<typename T, typename>

std::experimental::optional<T> Typed\_main\_binding<Args...>::get\_opt(int argc, char\* argv[])

{

constexpr auto opt\_template\_idx = index\_of<T, Args...>();

for (const auto& p : m\_found\_parameters)

{

if(p.first == opt\_template\_idx)

return std::experimental::optional<T>(T());

}

return std::experimental::nullopt;

}

template<typename... Args>

template<typename T, typename, typename>

std::experimental::optional<T> Typed\_main\_binding<Args...>::get\_opt(int argc, char\* argv[])

{

constexpr auto opt\_template\_idx = index\_of<T, Args...>();

for (const auto& p : m\_found\_parameters)

{

if (p.first == opt\_template\_idx)

{

T param;

if (p.second+1 >= argc)

throw std::invalid\_argument("No parameter after an option tag which need one");

param.value = parse<decltype(param.value), true>(argv[p.second+1]);

return std::experimental::optional<T>(param);

}

}

return std::experimental::nullopt;

}

template<typename... Args>

template<typename T, typename, typename, typename>

std::experimental::optional<T> Typed\_main\_binding<Args...>::get\_opt(int argc, char\* argv[])

{

constexpr auto opt\_template\_idx = index\_of<T, Args...>();

for (const auto& p : m\_found\_parameters)

{

if (p.first == opt\_template\_idx)

{

T tagless\_option\_param;

tagless\_option\_param.value = parse<decltype(tagless\_option\_param.value), true>(argv[p.second]);

return std::experimental::optional<T>(tagless\_option\_param);

}

}

// This kind of option is actually mandatory

throw std::invalid\_argument("Missing tagless mandatory parameter");

}

template<typename ... Args>

template<typename T, typename>

std::vector<std::string> Typed\_main\_binding<Args...>::get\_option\_tags() {

const auto& tags\_array = T::get\_option\_tags();

return std::vector<std::string>(std::begin(tags\_array), std::end(tags\_array));

}

template<typename ... Args>

template<typename T, typename, typename>

std::vector<std::string> Typed\_main\_binding<Args...>::get\_option\_tags() { return std::vector<std::string>(0); }

}

#endif

# Graph.h

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

Graph  - A graph with template nodes

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

date                 : 12/2015

copyright            : (C) 2015 by B3311

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

//-------------------- Interface de la classe template Graph<T> ------------------

#ifndef GRAPH\_H

#define GRAPH\_H

//--------------------------------------------------------------- Includes systeme

#include <memory>

#include <unordered\_map>

#include <initializer\_list>

namespace TP3

{

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

//! Classe représentant un graphe fait de noeuds contenant des données de type T

//! et de liens orientés entre ces noeuds. Un noeud a nescessairement au moins un

//! lien (noeuds ajoutés automatiquement lors de l'ajout de liens). Si un même lien

//! est ajouté plusieurs fois, alors son nombre d'occurrences associé est incrémenté.

//! @remarks

//! L'utilisateur doit surcharger la fonction std::hash pour le type T si elle

//! n'existe pas dejà.

template<typename T, typename Hash = std::hash<T>, typename KeyEqual = std::equal\_to<T>,

typename Allocator = std::allocator<T>>

class Graph

{

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

public:

//----------------------------------------------- Types et alias publiques

//! Alias pour le type permettant d'identifier un noeud

using node\_id\_t = unsigned int;

//! Structure utilisée pour représenter la destination et le nombre d'occurrences d'un lien

struct Link\_to

{

Link\_to() = default;

Link\_to(const Link\_to&) = default;

Link\_to& operator=(const Link\_to&) = default;

Link\_to(Link\_to&&) = default;

Link\_to& operator=(Link\_to&&) = default;

explicit Link\_to(node\_id\_t dest) : occurrence(1), destination(dest) { };

unsigned int occurrence = 0;

node\_id\_t destination;

};

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

//! Ajoute un lien au graph

//! @params source référence universelle vers la valeur du noeud source du lien

//! @params destination référence universelle vers la valeur de la destination du lien

void add\_link(T&& source, T&& destination);

//! Retourne une multimap non ordonée représentant les liens du graph et associant à l'identifiant

//! d'un noeud source, une structure contenant l'id du noeud de destination et le nombre d'occurrences du lien.

std::unordered\_multimap<node\_id\_t, Link\_to>& get\_links();

//! Retourne une map non ordonée associant la valeur de chaque noeud à leur identifiant

std::unordered\_map<T, node\_id\_t>& get\_nodes();

//! Vide le graphe de ses noeuds et liens

void clear();

//----------------------------------------------------- Méthodes spéciales

//! Constructeur par défaut ou permettant de spécifier la fonction de hashage, l'opérateur

//! d'égalité et l'allocateur pour le type T.

//! Par defaut, std::Hash<T>(), std::equal\_to<T>() et std::allocator<T>() sont utilisés.

explicit Graph(const Hash& hash = Hash(),

   const KeyEqual& equal = KeyEqual(),

   const Allocator& alloc = Allocator());

//! Constructeur permettant de spécifier une liste d'initialisation, la fonction de hashage,

//! l'opérateur d'égalité et l'allocateur pour le type T.

//! Par defaut, std::Hash<T>(), std::equal\_to<T>() et std::allocator<T>() sont utilisés.

//! @params init\_list list d'initialisation contenant des paires représentant des liens

//! (pair.fisrt est le noeud source et pair.second est la destination).

Graph(std::initializer\_list<std::pair<T, T>> init\_list,

  const Hash& hash = Hash(),

  const KeyEqual& equal = KeyEqual(),

  const Allocator& alloc = Allocator());

// On utilise l'implémentation par défaut du copy/move constructor et copy/move assignment operator

Graph(const Graph&) = default;

Graph& operator=(const Graph&) = default;

Graph(Graph&&) = default;

Graph& operator=(Graph&&) = default;

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

private:

//------------------------------------------------------- Attributs privés

//! attribut contenant le prochain id attribué à un nouveau noeud (pas indispensable ici, mais est

//! utile pour une implémenation future de méthodes de supression d'éléments)

node\_id\_t m\_new\_id = 0;

//! multimap non ordonnée stockant tout les liens entre les nœuds du graphe (les clés sont les

//! identifiants des nœuds source, les valeurs sont les destinations et leur nombre d'occurrences)

std::unordered\_multimap<node\_id\_t, Link\_to> m\_links;

//! map non ordonée contenant les noeuds (valeur et identifiant)

std::unordered\_map<T, node\_id\_t> m\_nodes;

};

//--------------------------------------------------------- Fonctions globales

//! Fonction créant un fichier GraphViz à partir d'un objet de type Graph<T>

//! @remarks utilise l'opérateur output stream (<<) sur le type T pour le serialiser

template<typename T>

void serialize\_graph(const std::string& output\_filename, std::unique\_ptr<Graph<T>> graph)

{

std::string line;

std::ofstream outfile(output\_filename, std::ios::trunc);

if (!outfile.is\_open())

throw std::invalid\_argument("Invalid output file path or don't have modification right");

outfile << "digraph {" << std::endl;

// Write nodes

for (const auto& node : graph->get\_nodes())

outfile << "node" << node.second << " [label=\"" << node.first << "\"];" << std::endl;

// Write links

for (const auto& link : graph->get\_links())

outfile << "node" << link.first << " -> "

<< "node" << link.second.destination

<< " [label=\"" << link.second.occurrence << "\"];" << std::endl;

outfile << "}" << std::endl;

outfile.close();

}

//--------------- Implémentation de la classe template Graph<T> --------------

template<typename T, typename Hash, typename KeyEqual, typename Allocator>

Graph<T, Hash, KeyEqual, Allocator>::Graph(const Hash& hash, const KeyEqual& equal, const Allocator& alloc)

: m\_links(), m\_nodes(10, hash, equal, alloc) { }

template<typename T, typename Hash, typename KeyEqual, typename Allocator>

Graph<T, Hash, KeyEqual, Allocator>::Graph(std::initializer\_list<std::pair<T, T>> init\_list,

const Hash& hash,

const KeyEqual& equal,

const Allocator& alloc)

: m\_links(), m\_nodes(10, hash, equal, alloc)

{

for (const auto& link : init\_list)

add\_link(link.first, link.second);

}

template<typename T, typename Hash, typename KeyEqual, typename Allocator>

void Graph<T, Hash, KeyEqual, Allocator>::add\_link(T&& source, T&& destination)

{

node\_id\_t source\_id, destination\_id;

// Add source node to m\_nodes if not stored yet and get source id

auto p = m\_nodes.emplace(std::make\_pair(std::forward<T>(source), m\_new\_id));

source\_id = p.second ? m\_new\_id++ : p.first->second; // TODO: verifier les incrémentations

// Add destination node to m\_nodes if not stored yet and get destination id

p = m\_nodes.emplace(std::make\_pair(std::forward<T>(destination), m\_new\_id));

destination\_id = p.second ? m\_new\_id++ : p.first->second;

// Find every links which have the same source

auto range = m\_links.equal\_range(source\_id);

// Linear lookup on resulting range to find if we have already added a link with the same destination and source

for (auto it = range.first; it != range.second; ++it)

{

if (it->second.destination == destination\_id)

{

++(it->second.occurrence); // increment link occurrence number

return;

}

}

// Store new link otherwise

m\_links.emplace(source\_id, Link\_to(destination\_id));

}

template<typename T, typename Hash, typename KeyEqual, typename Allocator>

inline std::unordered\_multimap<typename Graph<T, Hash, KeyEqual, Allocator>::node\_id\_t,

typename Graph<T, Hash, KeyEqual, Allocator>::Link\_to>&

Graph<T, Hash, KeyEqual, Allocator>::get\_links()

{

return m\_links;

}

template<typename T, typename Hash, typename KeyEqual, typename Allocator>

inline std::unordered\_map<T, typename Graph<T, Hash, KeyEqual, Allocator>::node\_id\_t>&

Graph<T, Hash, KeyEqual, Allocator>::get\_nodes()

{

return m\_nodes;

}

template<typename T, typename Hash, typename KeyEqual, typename Allocator>

inline void Graph<T, Hash, KeyEqual, Allocator>::clear()

{

m\_links.clear();

m\_nodes.clear();

m\_new\_id = 0;

}

}

#endif // GRAPH\_H

# Log\_parser.h

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

Log\_parser  - An apache log parser

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

date                 : 12/2015

copyright            : (C) 2015 by B3311

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

//----------------------- Interface de la classe Log\_parser ----------------------

#ifndef LOG\_PARSER\_H

#define LOG\_PARSER\_H

//-------------------------------------------------------------- Includes systèmes

#include <memory>

#include <string>

#include "optional.h"

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

#include "Graph.h"

namespace TP3

{

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

//! Classe permettant de parser un log apache et d'en déduire, soit un graphe

//! des requettes GET soit la liste des URLs utilisés dans ces requêtes

class Log\_parser

{

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

public:

//----------------------------------------------- Types et alias publiques

using hour\_t = unsigned short;

using URL\_t = std::string;

using graph\_t = Graph<URL\_t>;

using urls\_scores\_t = std::unordered\_multimap<URL\_t, unsigned int>;

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

void enable\_hour\_filter(hour\_t hour);

void disable\_hour\_filter();

void enable\_exclusion() noexcept;

void disable\_exclusion() noexcept;

//! Parse le log donné en parametre pour trouver la liste des URLs utlisés

//! lors de requêtes GET valides associés à leur nombre d'occurrences.

//! @remarks

//! - Si 'enable\_hour\_filter(hour\_t)' a été appellé, les résultat ne comprendra

//! que les requettes faites à l'heure spécifiée.

//! - Si enable\_exclusion() a été appellé, les résultat ne comprendra que les

//! requettes qui ne concernent pas des fichier css, javascript ou des images.

//! @throws std::invalid\_argument if specified log file couldn't be read

std::unique\_ptr<urls\_scores\_t> parse\_urllist(const std::string& log\_file\_name) const;

//! Parse le log donné en parametre pour en déduire le graphe des requêtes

//! GET valides.

//! @remarks

//! - Si 'enable\_hour\_filter(hour\_t)' a été appellé, les résultat ne comprendra

//! que les requettes faites à l'heure spécifiée.

//! - Si enable\_exclusion() a été appellé, les résultat ne comprendra que les

//! requettes qui ne concernent pas des fichier css, javascript ou des images.

//! @throws std::invalid\_argument if specified log file couldn't be read

std::unique\_ptr<graph\_t> parse\_graph(const std::string& log\_file\_name) const;

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

private:

//------------------------------------------------------- Méthodes privées

//! Fonction aidant à la lecture d'un log apache ligne par ligne

void for\_each\_log\_line(const std::string& log\_file\_name,

const std::function<void(URL\_t, URL\_t)>& parsing\_func) const;

//------------------------------------------------------- Attributs privés

bool m\_is\_exclusion\_filter\_enabled = false;

std::experimental::optional<hour\_t> m\_filter\_hour = std::experimental::nullopt;

};

}

#endif // LOG\_PARSER\_H

# Log\_parser.cpp

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

Log\_parser  - An apache log parser

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

date                 : 12/2015

copyright            : (C) 2015 by B3311

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

//-------------------- Implémentation de la classe Log\_parser --------------------

//-------------------------------------------------------------- Includes systèmes

#include <array>

#include <regex>

#include <string>

#include <memory>

#include <utility>

#include <fstream>

#include <iterator>

#include "optional.h"

//------------------------------------------------------------ Includes personnels

#include "Log\_parser.h"

#include "Graph.h"

#include "Utils.h"

namespace TP3

{

//------------------------------------------------------------------- STATIQUE

//------------------------------------------------------------ Regex statiques

//! Extentions to be ignored if media exclusion filter is enabled

static const std::string EXCLUDED\_EXTENTIONS = "js|css|png|ogg|jpg|jpeg|bmp|raw|tiff|gif|ppm|pgm|pbm|pnm|webp|heif|

bpgcd5|deep|ecw|fits|fits|flif|ILBM|IMG|Nrrd|PAM|PCX|PGF|PLBM|SGI|SID|TGA|VICAR";

//! Regex matching an IP (ipv4 or ipv6)

static const std::string IP\_REGEX = R"regex((?:(?:[0-9]{1,3}\.){3}|^(?:[0-9]{1,3}\.){5})[0-9]{1,3})regex";

//! Regex matching a timestamp (sub-matches hour and GMT+...)

static const std::string TIMESTAMP\_REGEX = R"regex((?:\[[0-9]{1,2}\/[A-Z][a-z]{2,3}\/[0-9]{4}:([0-9]{1,2}):(?:[0-

9]{1,2}:?){2} ((?:\+|\-)?[0-9]{4})\]))regex";

//! Regex matching an absolute URL (submatches insa intranet domain name if any) (based on regex of 'stephenhay'

//! in link bellow)

//! @link https://mathiasbynens.be/demo/url-regex

static const std::string ABSOLUTE\_URL\_REGEX = R"regex("((?:((?:https?|ftp):\/\/intranet-if\.insa-lyon\.fr)|

(?:(?:https?|ftp):\/\/[^\s/$.?#].))[^\s]\*)")regex";

//! Regex matching a GET request (submatches relative URL)

static const std::string GET\_REGEX = R"regex("GET\s+(\/[^\s]\*)\s+HTTPS?\/[0-9]+.[0-9]+")regex";

static const std::string GET\_REGEX\_E = R"regex("GET\s+(\/[^\s]\*\.(?:(?!)regex"

+ EXCLUDED\_EXTENTIONS + R"regex(|\s).)\*)\s+HTTPS?\/[0-9]+.[0-9]+")regex";

//! Regex matching data size number

static const std::string SIZE\_REGEX = R"regex([0-9]+)regex";

//! Regex matching an apache log line with GET and 200 as status code (hour and URLs can be extracted (submatches))

static const std::regex LOG\_LINE\_REGEX = std::regex(

  R"r(^\s\*)r" + IP\_REGEX + R"regex(\s+-\s+-)regex" // <IP> - -

+ R"r(\s+)r" + TIMESTAMP\_REGEX

+ R"r(\s+)r" + GET\_REGEX

+ R"regex(\s+200\s+[0-9]+)regex" // status code = 200 followed by data size number

+ R"r(\s+)r" + ABSOLUTE\_URL\_REGEX

+ R"r(\s+)r" + R"regex(\".+\"$)regex",

std::regex::icase | std::regex::optimize); // use '/i' flag in order to ignore case

//! Regex matching an apache log line with GET and 200 as status code (hour and URLs can be extracted (submatches))

static const std::regex LOG\_LINE\_REGEX\_E = std::regex(

  R"r(^\s\*)r" + IP\_REGEX + R"regex(\s+-\s+-)regex" // <IP> - -

+ R"r(\s+)r" + TIMESTAMP\_REGEX

+ R"r(\s+)r" + GET\_REGEX\_E

+ R"regex(\s+200\s+[0-9]+)regex" // status code = 200 followed by data size number

+ R"r(\s+)r" + ABSOLUTE\_URL\_REGEX

+ R"regex(\s+)regex" + R"regex(\".+\"$)regex",

std::regex::icase | std::regex::optimize); // use '/i' flag in order to ignore case

static const size\_t LINE\_MATCH\_NUM = 0;

static const size\_t HOUR\_MATCH\_NUM = 1;

static const size\_t TIMEZONE\_MATCH\_NUM = 2;

static const size\_t GET\_URL\_MATCH\_NUM = 3;

static const size\_t REFERER\_MATCH\_NUM = 4;

static const size\_t INSA\_INTRANET\_MATCH\_NUM = 5;

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

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

void Log\_parser::enable\_hour\_filter(hour\_t hour) { m\_filter\_hour = hour; }

void Log\_parser::disable\_hour\_filter() { m\_filter\_hour = std::experimental::nullopt; }

void Log\_parser::enable\_exclusion() noexcept { m\_is\_exclusion\_filter\_enabled = true; }

void Log\_parser::disable\_exclusion() noexcept { m\_is\_exclusion\_filter\_enabled = false; }

std::unique\_ptr<Log\_parser::urls\_scores\_t> Log\_parser::parse\_urllist(const std::string& log\_file\_name) const

{

// Multimap storing documents URLs with their occurrence number

auto urls = std::unique\_ptr<urls\_scores\_t>(new urls\_scores\_t()); // C++14: std::make\_unique<urls\_scores\_t>();

for\_each\_log\_line(std::move(log\_file\_name), [this, &urls](URL\_t doc\_url, URL\_t referer\_url)

{

auto url\_it = urls->find(doc\_url);

if (url\_it != std::end(\*urls))

url\_it->second++;

else

urls->emplace(std::move(doc\_url), 1);

});

return urls;

}

std::unique\_ptr<Log\_parser::graph\_t> Log\_parser::parse\_graph(const std::string& log\_file\_name) const

{

auto log\_graph = std::unique\_ptr<graph\_t>(new graph\_t()); // C++14: std::make\_unique<graph\_t>();

for\_each\_log\_line(std::move(log\_file\_name), [this, &log\_graph](URL\_t doc\_url, URL\_t referer\_url)

{

log\_graph->add\_link(std::move(referer\_url), std::move(doc\_url));

});

return log\_graph;

}

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

//----------------------------------------------------------- Méthodes privées

void Log\_parser::for\_each\_log\_line(const std::string& log\_file\_name,

const std::function<void(URL\_t, URL\_t)>& parsing\_func) const

{

std::string line;

std::ifstream infile(log\_file\_name);

if (!infile.is\_open())

throw std::invalid\_argument("Invalid log file name or don't have reading right (can't open log)");

while (std::getline(infile, line))

{

std::smatch matches;

if (std::regex\_search(line, matches, (m\_is\_exclusion\_filter\_enabled ? LOG\_LINE\_REGEX\_E : LOG\_LINE\_REGEX)))

{

// Apply hour filter if enabled

if (m\_filter\_hour) {

auto timezone = parse<int>(matches[TIMEZONE\_MATCH\_NUM].str()) / 100;

auto hour = parse<hour\_t>(matches[HOUR\_MATCH\_NUM].str());

if (\*m\_filter\_hour != static\_cast<hour\_t>(positive\_mod(hour - timezone, 24)))

continue;

}

auto doc\_url = matches[GET\_URL\_MATCH\_NUM].str();

const auto& intranet\_match = matches[INSA\_INTRANET\_MATCH\_NUM];

if (intranet\_match.matched) {

// Remove insa intranet domain name

auto referer\_url = matches[REFERER\_MATCH\_NUM].str().substr(intranet\_match.str().size());

parsing\_func(std::move(doc\_url), referer\_url);

}

else

parsing\_func(std::move(doc\_url), matches[REFERER\_MATCH\_NUM].str());

}

}

if (infile.bad())

throw std::invalid\_argument("Error while reading log file");

infile.close();

}

}

# Manuel d’utilisation

NAME

log\_analyzer - Analyze Apache Log

SYNOPSIS

log\_analyzer LOG\_FILE [OPTION]...

DESCRIPTION

Analyzes a given apache server LOG\_FILE to output a 'get' request graph or a list of most used URLs.

-g, -G, --graph

Creates a graph in which each node is a URL, and each link is a successfull

GET request between these nodes. Outputs GraphViz text file using specfied filename.

If this option isn't specified, the program will only output a list of most used URLs

with their respective occurrence number (sorted in descending usage order).

-l, -L, --listCount

Specifies the maximum number of URLs in the outputed list of most used URLs.

Must be greater than 1 (10 by default).

-e, -E, --excludeMedia

Excludes css, javascript and image files from the result.

-t, -T, --hour

Ignore all log entries which haven't been added at the specfied hour (hour GMT+0000).

-h

Show this documentation

EXAMPLES

log\_analyzer mylog.log --hour 12 -e

Get top ten list of URLs used at 12th hour and which aren't images, css nor javascript files

log\_analyzer mylog.log -e -g graph.dot

Get GET request graph of URLs which aren't images, css nor javascript files

log\_analyzer mylog.log -t 2 -l 20

Get top 20 list of URLs used at 2th hour

REPORTING BUGS

You didn't saw any bugs.