

# TP C++ n°4

## *Analyse de logs apache*

### 1. Introduction

Le web est basé sur un échange de données entre un client (le navigateur) et un serveur. Le client effectue une requête par l'intermédiaire du protocole HTTP (HyperText Transfer Protocol). Cette requête est transmise via le réseau vers le serveur. Le serveur fait en retour une réponse consistant notamment en un code retour, et un document. Lorsque cette réponse est un document HTML, le client a la possibilité de cliquer sur un lien s'y trouvant afin d'effectuer une autre requête, c'est le principe même de la navigation. Toutes ces opérations sont consignées au niveau du serveur web sous la forme d'un fichier journal (*log* en anglais).

Nous proposons dans ce TP d'analyser ces fichiers pour en tirer de l'information.

### 2. Structure du fichier

Un fichier journal, habituellement un .txt ou un .log, va consigner pour chaque page ou document vu sur un site, un certain nombre d'information (adresse IP, fichier atteint (hit), date / heure,...). Ces informations sont stockées sur une ligne. Chaque ligne représente un accès à l'un des documents du site web.

Les fichiers de log que nous allons utiliser sont ceux générés par Apache. Ils sont composés de lignes qui ont cette structure :

```
192.168.0.0 - - [08/Sep/2012 :11:16:02 +0200] "GET /temps/4IF16.html
HTTP/1.1" 200 12106 "http://intranet-if.insa-lyon.fr/temps/4IF15.html"
"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1"
```

**192.168.0.0**

L'adresse IP du client émetteur de la requête. Dans les logs qui vous seront fournis, cette donnée a été rendue anonyme.

- -

Le premier - remplace le User Logname, soit le nom d'utilisateur du visiteur.

Le second - remplace le nom d'utilisateur (ou Authenticated User) que l'internaute s'est donné lui-même.

Dans le cadre de ce TP, ces champs ont été rendus anonymes.

**[08/Sep/2012:11:16:02 +0200]**

La date et l'heure de la requête (l'heure exacte de la visite)

La différence en rapport avec l'heure de Greenwich (GMT)

**"GET /temps/4IF16.html HTTP/1.1"**

La requête http constituée de :

Le type d'action exécutée (GET, POST, OPTIONS, etc.)

La partie qui nous intéresse ici est /temps/4IF16.html qui représente l'URL (Uniform Resource Locator) du document demandé (l'objet associé à l'action). Cette adresse est relative par rapport à l'url de base du site.

**200**

Le *status* ou *return code* HTTP est un code retour de la réponse du serveur. 200 signifie que cela s'est bien passé.

**12106**

La quantité de données transférées au serveur distant pour accomplir l'opération, donc la taille en octets de la réponse.

**"http://intranet-if.insa-lyon.fr/temps/4IF15.html"**

Le *referer* (référant/référenceur), c'est-à-dire l'adresse sur laquelle le navigateur se trouvait lorsqu'il a effectué sa requête.

⚠ Il faudra enlever la base de l'url quand elle est locale !

**"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:14.0) Gecko/20100101 Firefox/14.0.1"**

L'identification du client navigateur.

### 3. Graphe de parcours

Étant donné qu'un utilisateur passe de page en page, son parcours peut être apparenté à un graphe où les nœuds sont les documents et les arcs représentent le passage d'un document à un autre. Les fichiers de log d'un seul serveur web ne pourront transcrire que les informations qui lui sont relatives.

Ces notions vous seront expliquées au tableau en début de séance.

## 4. Expression (incomplète) des besoins

Le TP consiste à concevoir et réaliser un outil en ligne de commande permettant d'analyser un fichier de log Apache. Cet outil sera capable de générer des documents synthétiques au format GraphViz (voir annexe), mais aussi des statistiques.

L'outil proposé prend en paramètres le nom du fichier de log ainsi qu'une série d'options en ligne de commande :

`$/analog [options] nomfichier.log`

Par défaut, c'est-à-dire quand il n'y a aucune option, il affichera sur la console sous forme textuelle la liste des 10 documents les plus consultés par ordre décroissant de popularité. Aucun fichier ".dot" n'est généré dans ce cas. Les options en ligne de commande sont les suivantes :

`[-g nomfichier.dot]`

Cette option permet de produire un fichier au format GraphViz du graphe analysé. Chaque document apparaîtra sous la forme d'un nœud et chaque arc indiquera le nombre de parcours associés.

`[-e]`

Cette option permet d'exclure tous les documents qui ont une extension de type image, css ou javascript.

`[-t heure]`

Cette option permet de ne prendre en compte que les hits qui sont dans le créneau horaire correspondant à l'intervalle [heure, heure+1[.

## 5. Méthode de travail

### A. Spécifications/conception

Tout ce qui est demandé dans cette partie doit être fait ***indépendamment du langage de programmation C++***.

- 1 **Spécifications.** Il faut dans un premier temps spécifier de manière complète le fonctionnement du programme. Pour cela, pensez à tous les cas de figure (cas normaux, cas d'erreur). Dans cette optique, il vous sera demandé un tableau récapitulatif des spécifications avec, pour chacune un ou plusieurs tests associés. Remarque : un *framework* de test vous est fourni, vous devez impérativement l'utiliser. Dans ce *framework*, l'identifiant du test est un nom de répertoire.

2 **Conception.** L'analyse préliminaire vous conduira à définir des classes composant l'application avec pour chacune son rôle, ses attributs et ses méthodes. Le choix de classes doit privilégier la **réutilisabilité**. Un schéma général de l'application avec les interactions entre classes devra être mis dans le compte-rendu.

3 **Données**<sup>1</sup>. La conception détaillée doit permettre de définir les structures de données employées pour représenter les attributs. Vous devrez donc réfléchir aux structures de données que vous aurez à utiliser. Ces structures de données devront être choisies par rapport aux fonctionnalités suivant les critères classiques d'utilisation de la mémoire et de performance des algorithmes associés. Vous justifierez ces deux aspects dans le compte-rendu. Vous présenterez dans ce compte-rendu un ou des schémas précis de l'implantation mémoire des structures de données choisies.

## B. Réalisation

L'un des objectifs du TP est la réutilisation de composants logiciels. On s'appuiera donc pour le codage, au maximum sur les classes (génériques ou non) fournies par la bibliothèque (STL). Par exemple, une première étape peut nécessiter d'écrire et de tester séparément une classe qui gèrera le fichier d'entrée. Une autre étape pourra être la réalisation d'une classe pour le stockage du graphe de parcours (ou plusieurs classes). Les tests fonctionnels de l'application devront être implémentés dans le *framework* fourni. Ils devront couvrir la totalité des spécifications. Dans le cas où votre programme ne fonctionnerait pas comme prévu par le cahier des charges, vous tenterez d'expliquer pourquoi.

## 6. Compte-rendu

Le compte rendu est composé de deux éléments séparés :

Un livrable électronique : le code source (\*.h, \*.cpp, makefile et arborescence de fichiers de tests<sup>2</sup>) de votre programme à déposer sur Moodle dans le répertoire : Accueil / ► Mes cours / ► Informatique / ► Informatique Apprentissage / ► IFA-3 / ► IFA-3-POO2 / ► TP4-C++-3IFA / ► Rendu du TP 4 de C++: Analyse de logs Apache (3IF A) avec **B35XX.zip** où **XX** : votre numéro de binôme

1 **Un compte-rendu papier contenant**

- ~~Un document de conception qui comprend : les spécifications générales de votre application, les spécifications plus détaillées avec pour chacune le (s) lien(s) vers le(s) test(s) fonctionnel(s) associé(s)<sup>3</sup>, l'architecture globale de l'application avec les classes principales, la présentation des structures de données commentées et justifiées ainsi que les schémas de structure.~~
- ~~Vos codes sources imprimés (.h et .cpp agrafés par classe et le programme principal)~~

---

<sup>1</sup> **Attention** : dans cette partie, on ne parle que d'algorithmes, de structure de données abstraites et pas encore de réalisation (STL) ! Aussi, on n'utilisera que les tableaux, les listes chaînées, ou arbres (et donc pas leur modélisation avec la STL).

<sup>2</sup> Ne mettez pas le fichier anonyme.log dans votre arborescence, si vous devez l'utiliser, référencez /tmp/anonyme.log

<sup>3</sup> On ne vous demande pas de tests unitaires pour les classes (même si ces tests sont réalisés).

- 2 Un manuel d'utilisation qui explique à quoi sert votre outil, de quelle façon l'appeler. Vous pouvez vous inspirer de la commande `man` d'UNIX.

**N.B.** Le compte rendu de ce T.P. ne devra pas excéder 6 pages, (hors codes sources et hors manuel d'utilisation) que vous présenterez selon les conventions d'écriture du Guide de style C++ INSA/IF (indentation, commentaires pour les zones critiques...).

## 7. Fichier de log test

Un fichier de log anonyme est à votre disposition pour des tests. Ce fichier étant assez volumineux (22 Mo), il convient d'en faire une copie locale dans `/tmp`, cela évitera de surcharger le réseau lors de l'exécution de votre programme :

soit	\$ cp \${HOME}/partage_public/tp/tp-log/anonyme.log /tmp (si le repertoire partage_public existe)
soit	\$ cp /shares/public/tp/tp-log/anonyme.log /tmp

\$ chmod 666 /tmp/anonyme.log (pour que le prochain binôme qui travaillera sur la même machine puisse y accéder lui aussi)

Chaque appel à votre exécutable devra donc explicitement donner le chemin du fichier :

\$ ./analog /tmp/anonyme.log

**Note :** Le partage\_public doit être présent dans votre répertoire de connexion pour que ce chemin d'accès soit valide.

## 8. Framework de test

Un *framework* de test est fourni dans :


\${HOME}/partage_public/tp/tp-log/Tests.tgz.
/shares/tp/tp-log/Tests.tgz
Accueil / ► Mes cours / ► Informatique / ► Informatique Apprentissage / ► IFA-3 / ► IFA-3-P002 / ► TP4-C++-3IFA / ► TP4 C++- STL : Fichiers Fournis

Ce *framework* permet d'exécuter une ligne de commande, d'en vérifier le code retour, la sortie produite et les fichiers produits. Il se présente sous la forme de scripts shell. Vous devrez l'utiliser dans le cadre de vos tests. Vous pouvez construire des fichiers de log de test moins volumineux que celui fourni qui permettront d'avoir une granularité de test plus fine et une plus grande facilité de prévision du résultat. Chaque test est associé à un répertoire qui contient de manière optionnelle une série de vérifications (un fichier d'aide est contenu dans le *framework*, merci de vous y référer). Un test comporte une ligne de commande que vous devrez indiquer de manière relative (exemple : `../bin/analog`). Si vous souhaitez effectuer un test sur `anonyme.log`, donnez comme argument le fichier `/tmp/anonyme.log`. Vous pourrez placer vos propres fichiers de log dans le répertoire de test directement.

## Annexe - exemples d'utilisation

Voici quelques exemples d'utilisation du programme avec le jeu d'essai court.log suivant :

```
192.168.0.1 - - [08/Sep/2012:11:15:00 +0200] "GET /page2.html HTTP/1.1" 200
2000 "http://intranet-if.insa-lyon.fr/page1.html" "Mozilla/5.0"
192.168.0.1 - - [08/Sep/2012:11:37:00 +0200] "GET /page1.html HTTP/1.1" 200
1000 "http://intranet-if.insa-lyon.fr/page2.html" "Mozilla/5.0"
192.168.0.1 - - [08/Sep/2012:12:12:00 +0200] "GET /page2.html HTTP/1.1" 200
2000 "http://intranet-if.insa-lyon.fr/page3.html" "Mozilla/5.0"
192.168.0.1 - - [08/Sep/2012:12:16:00 +0200] "GET /page3.html HTTP/1.1" 200
3000 "http://intranet-if.insa-lyon.fr/page2.html" "Mozilla/5.0"
192.168.0.1 - - [08/Sep/2012:12:16:01 +0200] "GET /image.jpg HTTP/1.1" 200
50000 "http://intranet-if.insa-lyon.fr/page3.html" "Mozilla/5.0"
192.168.0.1 - - [08/Sep/2012:12:59:00 +0200] "GET /page2.html HTTP/1.1" 200
2000 "http://intranet-if.insa-lyon.fr/page1.html" "Mozilla/5.0"
```

 Attention, ces exemples ne se veulent pas exhaustifs et ne couvrent pas du tout la totalité des spécifications que vous aurez à rédiger.

---

### 1. Exécution sans option

```
$ ./analog court.log
/page2.html (3 hits)
/image.jpg (1 hits)
/page3.html (1 hits)
/page1.html (1 hits)
```

---

### 2. Exécution avec les options -e et -g

```
$ ./analog -e -g court.dot court.log
Dot-file court.dot generated
/page2.html (3 hits)
/page3.html (1 hits)
/page1.html (1 hits)
```

Le fichier court.dot généré :

```
digraph {
node1 [label="/page1.html"];
node0 [label="/page2.html"];
node2 [label="/page3.html"];
node0 -> node1 [label="1"];
node0 -> node2 [label="1"];
node1 -> node0 [label="2"];
node2 -> node0 [label="1"];
}
```



Pour générer l'image :

```
$ dot -Tpng -o court.png court.dot
```

---

### 3. Exécution avec les options -t et -g

```
$ ./analog -t 12 -g court.dot court.log
```

Dot-file court.dot generated

Warning : only hits between 12h and 13h have been taken into account

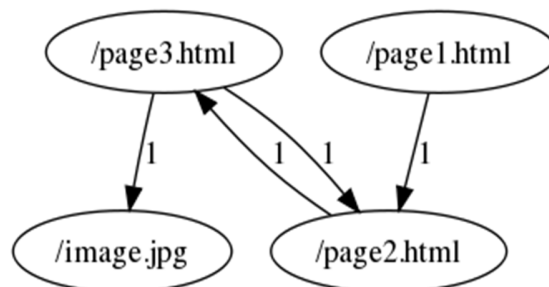
/page2.html (2 hits)

/image.jpg (1 hits)

/page3.html (1 hits)

Le fichier court.dot généré :

```
digraph {
node3 [label="/image.jpg"];
node1 [label="/page1.html"];
node0 [label="/page2.html"];
node2 [label="/page3.html"];
node0 -> node2 [label="1"];
node1 -> node0 [label="1"];
node2 -> node0 [label="1"];
node2 -> node3 [label="1"];
}
```



## Annexe - GraphViz

GraphViz est un outil qui permet à partir d'une description textuelle d'un graphe (nœuds, arcs et attributs associés) de créer un agencement des différents éléments et de produire en sortie une image.

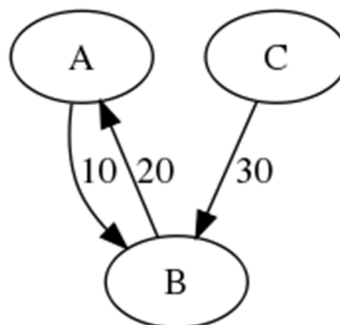
Voici un exemple simple de source décrivant un graphe :

```
digraph {  
    A;  
    B;  
    C;  
    A -> B [label="10"];  
    B -> A [label="20"];  
    C -> B [label="30"];  
}
```

Pour lancer la création du graphe au format png par exemple :

```
$ dot -Tpng -o out.png in.dot
```

qui produira l'image suivante :



Il existe toute une série de propriétés associées au graphe, aux nœuds ainsi qu'aux arcs que l'on peut indiquer de manière optionnelle. La liste exhaustive se trouve ici :

<http://www.graphviz.org/doc/info/attrs.html>