

## Objectifs

- Réaliser l'itération 4 de l'application Gestion Log
  - Ouvrir un fichier texte
  - Extraire les informations d'un fichier texte

## Contenu technique

- Ouverture, lecture et fermeture d'un fichier

## Durée 6h itération, 2h exercices (à faire à la maison)

Il est temps de passer à la seconde partie de notre application, le filtrage de logs et la sauvegarde des logs

Cette application va être réalisée en 3 séances de 6h.

- La première séance sera consacrée à l'ouverture et à l'extraction de données depuis un fichier.
- La seconde séance consistera à extraire et sauvegarder les données d'un autre fichier.
- La dernière séance consistera à envoyer les logs sur le serveur centralisé.

# I. Utilisation (plus) avancée de Git et configuration de votre projet

## I.1 Branches dev et master

Nous allons améliorer la gestion de notre projet avec git.

Pour cela,

- Vous allez créer une branche **dev** qui servira aux développements.
- Lorsque celui-ci sera terminé, vous fusionnerez votre branche **dev** avec la branche **master**
- Vous publierez un **tag** sur la branche master à la fin de chaque itération.

Tâche1. Créer une branche dev et basculer dessus

## I.2 Module log

Vous allez créer un nouveau module (`log.h` et `log.cpp`) qui accueillera vos nouvelles fonctions.

Tâche2. Ajouter le module comme lors de l'itération 3



Tâche3. Modifier le `CmakeList.txt` de votre projet pour intégrer la bibliothèque

```
#Extrait du fichier CmakeList.txt
#Ajouter les fichiers sources de votre projet
set(SRCS
    Logs/log.cpp
)
#Ajouter les fichiers headers de votre projet
set(HEADERS
    Logs/log.h
)
include_directories(Logs)
...
```

## I.3 Outils qualités

Tâche4. Documenter et formater votre code.

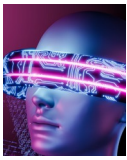
Tâche5. Pousser votre travail sur le dépôt GIT, branche **dev**

Résultats attendus :

- ✓ La branche dev est créée et c'est la branche active du projet
- ✓ Le module Logs est correctement créé

## I.4 Validation

**Appeler l'enseignant pour qu'il valide votre travail**



## II. Ouverture, lecture et fermeture d'un fichier

Tâche6. Consulter la page de manuel de `ofstream`, `ifstream`, `getline()`, `close()`

Q1. Quel est le rôle de ces fonctions ?

### II.1 Manipuler un fichier texte

*Avant de manipuler les fichiers de logs du projet, vous allez tester l'ouverture, l'écriture et la fermeture d'un fichier en C++.*

#### II.1.1 Ouverture et écriture d'un fichier texte

- Créer une fonction `testEcritureFichier()` et réaliser les manipulations suivantes :

Tâche7. Ouvrir en langage C++ un nouveau fichier appelé `test.txt`

Tâche8. Écrire des informations (plusieurs lignes svp) à l'intérieur du fichier.

Tâche9. Ouvrir le fichier avec un éditeur et vérifier que les informations écrites sont bien présentes dans le fichier.

#### II.1.2 Ouverture et lecture d'un fichier

- Créer une fonction `testLectureFichier()` et réaliser les manipulations suivantes :

Tâche10. En C++, ouvrir dans une nouvelle fonction le fichier précédent et lire son contenu.

Q2. Comment avez-vous lu chacune des lignes jusqu'à la fin du fichier ?

#### II.1.3 Fermeture du fichier

Tâche11. Déterminer la fonction C++ responsable de la fermeture du fichier à la fin du traitement.

Tâche12. Intégrer cette fonction dans votre code (si ce n'est déjà fait)

#### II.1.4 Versionning

- Documenter et formater votre code.
- Pousser votre travail sur le dépôt GIT, branche **dev**

Résultats attendus :

- ✓ Les 2 fonctions sont créées dans le module Logs
- ✓ Un fichier contenant plusieurs lignes est créé et lu depuis le programme.

## II.2 Validation

**Appeler l'enseignant pour qu'il valide votre travail**



## II.3 Fonction sudoLog ( )

Tâche13. Étudier la documentation suivante

<https://www.malekal.com/comment-lire-les-logs-sur-linux-en-temps-reel-avec-tail-multitail/>

### Extrait de la documentation

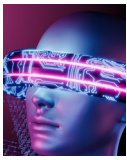
Dans Linux les journaux système se trouvent dans **/var/log/**, on y trouve auth.log, kern.log, messages, syslog, daemon.log qui sont des journaux du système Linux.

- **/var/log/boot.log** : Journal de démarrage du système (le journal de démarrage stocke toutes les informations relatives aux opérations de démarrage)
- **/var/log/auth.log** : journaux d'authentification (le journal d'authentification stocke tous les journaux d'authentification, y compris les tentatives réussies et échouées)
- **/var/log/debug** : journaux de débogage (le journal de débogage stocke des messages détaillés liés au débogage et est utile pour dépanner des opérations système spécifiques)
- **/var/log/daemon.log** : journaux des démons (le journal des démons contient des informations sur les événements liés à l'exécution de l'opération Linux)
- **/var/log/maillog** : journaux du serveur de messagerie (le journal de messagerie stocke les informations relatives aux serveurs de messagerie et à l'archivage des e-mails)
- **/var/log/kern.log** : Journaux du noyau (le journal du noyau stocke les informations du noyau Ubuntu Linux)
- **/var/log/btmp** : enregistrements d'échecs de tentatives de connexion
- **/var/log/utmp** : état de connexion actuel, par utilisateur
- **/var/log/wtmp** : historique des connexions / déconnexions
- **/var/log/lastlog** : informations sur les dernières connexions pour tous les utilisateurs. Ce fichier binaire peut être lu par la commande lastlog.

*Lorsque qu'un utilisateur veut utiliser temporairement une commande qui nécessite des droits super utilisateur (administrateur) sous Linux, il fait précéder sa commande de `sudo` (super user do) pour demander une demande d'approbation temporaire des droits administrateur.*

### II.3.1 Lecture des logs

Q3. Déterminer quel fichier de log sous Linux contient les informations sur les demandes d'approbations temporaire de droits administrateur.



Tâche14. Ouvrir dans votre projet C++ le fichier contenant les entrées `sudo`, et écrire les informations de ce fichier dans la console.

Remarque : Par défaut votre utilisateur n'a pas les droits de lecture sur le fichier contenant les logs. Il faut donc ajouter votre utilisateur au groupe `adm` pour pouvoir lire le fichier `auth.log` dans votre programme.

Tâche15. Pour ajouter votre utilisateur au groupe `adm`, exécuter les commandes suivantes

```
sudo usermod -aG adm $USER  
newgrp adm
```

## II.3.2 Filtrer les logs

Ce fichier ne contient pas que les informations concernant `sudo`, vous allez devoir n'afficher dans votre programme que les entrées `sudo` en filtrant les informations contenues dans ce fichier.

Tâche16. Filtrer dans votre projet C++ les informations pour n'afficher que les entrées correspondant à `sudo`

## II.3.3 Versionning

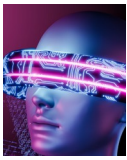
- Documenter et formater votre code.
- Pousser votre travail sur le dépôt GIT, branche **dev**

Résultats attendus :

- ✓ Le contenu du fichier `auth.log` est affichée dans la console lors de la sélection de la bonne entrée du menu (1-Afficher les logs `sudo`)
- ✓ Le contenu du fichier `auth.log` est filtrée pour n'afficher QUE les entrées correspondant à `sudo`

## II.3.4 Validation

- **Appeler l'enseignant pour qu'il valide votre travail**



## III. Outils DevOps

### III.1 Sauvegarde sur Git

Il est temps de sauvegarder votre travail.

Cette itération étant terminée et votre fonctionnalité réalisée, vous pouvez maintenant fusionner les branches **dev** et **master**.

La procédure est détaillée sur la page [Git, workflow basique des projets](#) du site de ressources.

<TL ;DR> En synthèse, vous devez réaliser les opérations ci-dessous.

- Sur la branche de dev (`commit` **et** `push`)
- Sur la branche master (`merge dev` **et** `push`)
- Créer un tag et pousser le tag

Résultats attendus :

- ✓ La branche dev est fusionnée avec master
- ✓ Les branches sont poussées sur le serveur Gitlab
- ✓ Un tag correspondant à l'itération est créé



## IV. Exercices sur les structures

Pour la prochaine itération, vous aurez besoin de manipuler un nouvel outil, les structures. Des exercices vous sont proposés. Ils sont à faire à la maison, sauf si vous êtes en avance sur le planning de l'itération.

Les exercices sont à faire et à rendre à la suite de votre compte rendu sur Moodle.

### IV.1 Réaliser les exercices suivants sur les structures

En vous aidant des informations sur le site de ressources

<https://www2.ciel-kastler.fr/docs/Programmation/Langage-c++/cahier-exercices-procedural.html#les-structures>

#### IV.1.1 Exercice 1- Stockage d'informations personnelles

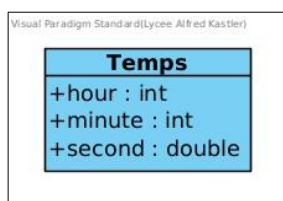
Réaliser deux versions du programme « stockage d'informations personnelle »

Q4. Version 1 – Sans utiliser de structure

Q5. Version 2 – En utilisant une structure.

#### IV.1.2 Exercice 2, Une structure Temps

Une structure Temps contient les champs suivants :



Q6. Réaliser le programme permettant de rentrer l'heure courante dans la structure puis de l'afficher.



### IV.1.3 Exercice 3, structures imbriquées (composition)

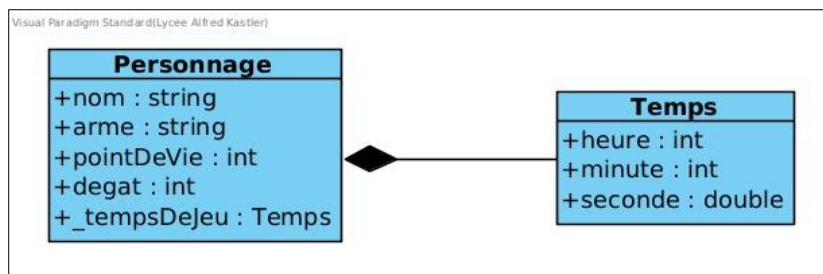
Les structures peuvent elle-mêmes contenir un champ de type Structure.

C'est ce qu'on appelle les structures imbriquées.

Pour illustrer ce concept, imaginons un jeu où chaque personnage à un nom, une arme (bâton, épée, baguette magique,...), des points de vie et inflige des dégâts.

Nous voulons également connaître le temps effectif de jeu de chaque personnage.

Nous obtenons la conception suivante :



Q7. Réaliser le programme permettant d'initialiser des personnages (tous les champs, y compris ceux de la structure imbriquée) et d'afficher les champs à l'écran.

Voici quelques exemples de personnages :

#### Personnage1

Nom	Gandalf
Arme	Baguette magique
Point de Vie	400
Dégât	25
_tempsDeJeu	50h 25min 04sec

#### Personnage2

Nom	Erik le nécromancien
Arme	épée
Point de Vie	100
Dégât	35
_tempsDeJeu	09h 39min 14sec





## V. Livrable

Sur le moodle de la section

→ Votre compte rendu de TP avec les réponses aux questions de celui-ci

Sur le serveur GIT de la section

→ Le projet VOTRE\_NOM\_Gestion\_Log

- branches master et dev
- tag v4.0