Rapport du Projet de Structure de données :

LU2IN006

Bengana Massyl

Munsup Warintara

Sujet:

Le but de ce projet est d'analyser un logiciel de gestion de versions en se focalisant sur les structures de données nécessaires pour permettre la création d'enregistrements instantanés du projet, la navigation entre les versions, la gestion d'une arborescence de versions, la vérification de l'identité des utilisateurs et la sauvegarde des modifications non enregistrées

Structures utilisées :

```
typedef struct cell {
 char * data ;
 struct cell * next ; } Cell ;
typedef Cell * List ;
```

Cette structure de données crée une liste chaînée simple. La structure "Cell" représente un élément individuel de la liste, et contient un pointeur vers une chaîne de caractères (data) et un pointeur vers l'élément suivant dans la liste (next). Le typedef suivant, qui définit "List" comme un pointeur vers un élément de type "Cell", permet d'utiliser "List" comme un alias pour ‘Cell * ‘.

```
typedef struct {
char * name ;
 char * hash ;
```

```
int mode ; } WorkFile ;

typedef struct {

WorkFile * tab ;

 int size ;

 int n ; } WorkTree ;
```

La structure "WorkFile" représente un fichier individuel et contient une chaîne de caractères pour le nom du fichier (name), une chaîne de caractères pour le hachage du fichier (hash) et un entier pour le mode du fichier (mode). La structure "WorkTree" représente une collection de fichiers, ou un arbre de fichiers, et contient un tableau de pointeurs vers des structures "WorkFile" (tab), un entier pour la taille du tableau (size) et un entier pour le nombre actuel de fichiers dans l'arbre (n).

```
typedef struct key_value_pair {

char * key ;

char * value ; } kvp ;

 typedef struct hash_table {

 kvp ** T ;

 int n ;

 int size ; } HashTable ;

typedef HashTable Commit ;
```

Cette structure de données crée une table de hachage avec des paires clé-valeur. La structure "kvp" représente une paire clé-valeur et contient deux chaînes de caractères, une pour la clé (key) et une pour la valeur (value). La structure "HashTable" représente une table de hachage et contient un tableau de pointeurs vers des paires clé-valeur (T), un entier pour le nombre actuel de paires dans la table (n) et un entier pour la taille de la table (size). La structure "Commit" est un alias pour "HashTable", donc "Commit" est simplement un autre nom pour "HashTable".

Description globale du code :

Le code de ce projet est organisé en plusieurs fichiers qui représentent les différentes parties du projet, ainsi qu'un fichier "myGit.c" qui sert de point d'entrée général. Cette division en fichiers permet de séparer les différentes fonctionnalités du projet et de les organiser de manière modulaire pour une meilleure maintenabilité et lisibilité du code. En outre, le projet comprend également des fichiers d'en-tête qui contiennent les déclarations de fonctions, les définitions de structures de données et d'autres éléments nécessaires à la compilation et à l'exécution du code. Cette approche modulaire facilite la gestion et la compréhension du projet dans son ensemble, en permettant une

séparation claire des responsabilités entre les différentes parties du code. A noté que l'on a un second main qui nous permet de tester les fonctions de la partie 1 a 5.

Fonctionnalités :

Le programme implémente un sous-ensemble des fonctionnalités de Git, un système de gestion de versions décentralisé. Voici les différentes fonctionnalités implémentées dans le programme :

• init : initialise le référentiel myGit, en créant les dossiers nécessaires et en initialisant les fichiers de références et de configuration.

• refs-list : affiche la liste des références présentes dans le dossier ".refs" du dossier "bak".

• create-ref : crée une nouvelle référence pour un commit donné.

• delete-ref : supprime une référence donnée.

• add : ajoute des fichiers `a la zone de préparation .

• clear-add : vide la zone de préparation.

• status : affiche le contenu de la zone de préparation.

• commit : crée un nouveau commit avec les fichiers présents dans la zone de préparation.

• get-current-branch : affiche la branche courante.

• branch : crée une nouvelle branche `a partir de la branche courante.

• branch-print : affiche les commits d'une branche donnée.

• checkout-branch : passe `a une autre branche.

• checkout-commit : passe `a un commit spécifique.

• merge : fusionne deux branches et gère les conflits ´éventuels