# Guide Développeur pour le Projet

Ce guide est destiné à fournir une compréhension approfondie des fonctionnalités du projet et des bonnes pratiques de développement.

# Introduction

Le projet consiste en la création d'un système de fichiers simulé en utilisant des structures de données et des opérations de fichiers en C. Le système de fichiers est conçu pour permettre la manipulation de fichiers, y compris la création, l'écriture, la lecture et l'ouverture de fichiers.

# Structure du Projet

Le projet est divisé en deux fichiers principaux :

- `projet.h`: Ce fichier contient les déclarations des fonctions, structures et constantes utilisées dans le projet.
- `projet.c`: Ce fichier contient les définitions des fonctions déclarées dans `projet.h`.

# Utilisation des Fonctions

Le fichier `projet.h` expose les fonctions suivantes :

- `int myFormat(char* partitionName) :` Cette fonction permet de formater une partition en initialisant les informations nécessaires pour le système de fichiers.
- `file* myOpen(char* fileName) :` Cette fonction permet d'ouvrir un fichier existant ou de créer un nouveau fichier s'il n'existe pas déjà.
- `int myWrite(file* f, void* buffer, int nBytes) :` Cette fonction permet d'écrire des données dans un fichier ouvert.
- `int myRead(file* f, void* buffer, int nBytes) :` Cette fonction permet de lire des données depuis un fichier ouvert.
- `void mySeek(file* f, int offset, int base) :` Cette fonction permet de déplacer le pointeur à l'endroit voulu.
- `void printHelp() :`Cette fonction affiche l'aide sur l'utilisation du programme.
- `void deleteFile() :` Cette fonction permet la suppression d'un fichier.
- `int deleteFileFromPartition(char* fileName) :` Cette fonction travaille avec deleteFile qui permet la supression du fichier dans la partition.
- `char** listFiles() :` Cette fonction permet d'afficher les fichiers existants.

- `int numFiles(char** files) :` Cette fonction permet de récupérer la longueur de la liste des fichiers
- `void deletePartition(char* partionName) :` Cette fonction permet de supprimer la partition à la fin du programme.

# Algorithme et Structures de Données Utilisés

## Algorithme

Le projet utilise un système de fichiers simulé qui repose sur les principes de la gestion de blocs de données et d'inodes. Voici un aperçu de l'algorithme utilisé :

1. **Formatage de la Partition**:
    - Lorsque la fonction `myFormat` est appelée, elle crée un fichier de partition, initialise les informations de super fichier et marque tous les blocs de données comme libres.
2. **Ouverture de Fichier**:
    - Lorsqu'un fichier est ouvert à l'aide de la fonction `myOpen`, l'algorithme recherche un inode associé au fichier. S'il n'existe pas, il en crée un nouveau. Il associe également un bloc de données libre au fichier.
3. **Écriture dans un Fichier**:
    - Lorsqu'un fichier est écrit à l'aide de la fonction `myWrite`, les données sont écrites dans les blocs de données associés au fichier. Si nécessaire, de nouveaux blocs de données sont alloués pour accueillir les données.
4. **Lecture depuis un Fichier**:
    - Lorsqu'un fichier est lu à l'aide de la fonction `myRead`, les données sont lues à partir des blocs de données associés au fichier.
5. **Suppression d’un fichier**:
    - Lorsqu'un fichier est supprimé à l’aide de la fonction `deleteFile`, le fichier est supprimé de la partition donc on libère tout l’espace utilisé par ce fichier.
6. **Suppression de la partition**:
    - Lorsque l’utilisateur quitte le programme `deletePartition` ferme le descripteur de fichier de la partition, libère la mémoire associée aux fichiers et blocs de données, puis supprime le fichier de partition spécifié.

## Structures de Données

Le projet utilise plusieurs structures de données pour gérer les fichiers et les blocs de données :

1. **Structure `DataBlock`**:
    - Cette structure représente un bloc de données et contient les données stockées dans le bloc, l'état du bloc (libre ou occupé) et un pointeur vers le bloc de données suivant.
2. **Structure `file`**:
    - Cette structure représente un fichier et contient le nom du fichier, sa taille, et la position actuelle dans le fichier.
3. **Structure `inode`**:
    - Cette structure représente un inode et contient le nom du fichier associé, un pointeur vers la structure de fichier, et un pointeur vers le premier bloc de données du fichier.
4. **Structure `SuperFileData`**:
    - Cette structure contient les données du super fichier, y compris le nombre d'inodes, la taille de la partition, un tableau d'inodes, un descripteur de fichier de la partition, la position actuelle dans la partition, et un tableau de blocs de données.

# Guide pour Améliorer/Modifier la Bibliothèque

Voici quelques conseils pour améliorer ou modifier la bibliothèque de système de fichiers :

1. **Optimisation de la Gestion des Blocs de Données**:
    - Vous pouvez explorer des algorithmes plus efficaces pour gérer l'allocation et la libération des blocs de données afin d'améliorer les performances du système de fichiers.
2. **Gestion des Erreurs Améliorée**:
    - Ajoutez des mécanismes de gestion des erreurs plus robustes et des messages d'erreur plus informatifs pour faciliter le débogage et la maintenance du code.
3. **Extension des Fonctionnalités**:
    - Explorez des fonctionnalités supplémentaires telles que la suppression de fichiers, le déplacement de fichiers, les permissions de fichiers, etc., pour rendre le système de fichiers plus complet.
4. **Amélioration de l'Interface Utilisateur**:
    - Améliorez l'interface utilisateur en ajoutant des fonctionnalités conviviales telles que la saisie sécurisée des noms de fichiers, la validation des entrées utilisateur, etc.

# Conclusion

Ce guide du développeur fournit une vue d'ensemble de l'algorithme et des structures de données utilisés dans le projet de système de fichiers. Il offre également des conseils pour améliorer et modifier la bibliothèque afin de répondre à divers besoins et exigences. En suivant ces conseils et en comprenant l'algorithme sous-jacent, les programmeurs seront en mesure d'apporter des modifications efficaces et pertinentes au projet.