**DATASTRUCTURES PROJECT**

**GITHUB**

**CODE:**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Repository;

class User;

class Commit {

private:

string message;

string author;

string timestamp;

public:

Commit(string \_message, string \_author, string \_timestamp)

: message(\_message), author(\_author), timestamp(\_timestamp) {}

string getMessage() const {

return message;

}

string getAuthor() const {

return author;

}

string getTimestamp() const {

return timestamp;

}

};

class File {

private:

string name;

string content;

public:

File(string \_name, string \_content) : name(\_name), content(\_content) {}

string getName() const {

return name;

}

string getContent() const {

return content;

}

};

class FileNode {

private:

File\* file;

vector<FileNode\*> children;

public:

FileNode(File\* \_file) : file(\_file) {}

File\* getFile() const {

return file;

}

const vector<FileNode\*>& getChildren() const {

return children;

}

void addChild(FileNode\* child) {

children.push\_back(child);

}

};

class Repository {

private:

string name;

bool isPublic;

User\* owner;

vector<Commit\*> commits;

FileNode\* root; // Root of the file tree

public:

Repository(string \_name, bool \_isPublic, User\* \_owner)

: name(\_name), isPublic(\_isPublic), owner(\_owner), root(nullptr) {}

string getName() const {

return name;

}

bool getVisibility() const {

return isPublic;

}

User\* getOwner() const {

return owner;

}

void addCommit(Commit\* commit) {

commits.push\_back(commit);

}

const vector<Commit\*>& getCommits() const {

return commits;

}

void addFile(File\* file, const string& path) {

if (!root) {

root = new FileNode(nullptr); // Root node is empty

}

FileNode\* currentNode = root;

vector<string> directories = splitPath(path);

for (const auto& dir : directories) {

FileNode\* childNode = findChildNode(currentNode, dir);

if (!childNode) {

File\* placeholder = new File(dir, ""); //

childNode = new FileNode(placeholder);

currentNode->addChild(childNode);

}

currentNode = childNode;

}

currentNode->addChild(new FileNode(file));

}

File\* getFile(const string& path) {

FileNode\* node = getNodeByPath(path);

if (node && node->getFile()) {

return node->getFile();

}

return nullptr;

}

private:

vector<string> splitPath(const string& path) {

vector<string> directories;

string directory;

for (char c : path) {

if (c == '/') {

if (!directory.empty()) {

directories.push\_back(directory);

directory.clear();

}

} else {

directory += c;

}

}

if (!directory.empty()) {

directories.push\_back(directory);

}

return directories;

}

FileNode\* findChildNode(FileNode\* parentNode, const string& name) {

for (FileNode\* child : parentNode->getChildren()) {

if (child->getFile()->getName() == name) {

return child;

}

}

return nullptr;

}

FileNode\* getNodeByPath(const string& path) {

vector<string> directories = splitPath(path);

FileNode\* currentNode = root;

for (const auto& dir : directories) {

currentNode = findChildNode(currentNode, dir);

if (!currentNode) {

return nullptr;

}

}

return currentNode;

}

};

class User {

private:

string username;

string password;

vector<Repository\*> repositories;

vector<User\*> followers;

public:

User(string \_username, string \_password) : username(\_username), password(\_password) {}

string getUsername() const {

return username;

}

string getPassword() const {

return password;

}

void addFollower(User\* follower) {

followers.push\_back(follower);

}

void removeFollower(User\* follower) {

for (auto it = followers.begin(); it != followers.end(); ++it) {

if (\*it == follower) {

followers.erase(it);

break;

}

}

}

const vector<User\*>& getFollowers() const {

return followers;

}

void createRepository(string repoName, bool isPublic) {

repositories.push\_back(new Repository(repoName, isPublic, this));

}

void deleteRepository(string repoName) {

for (auto it = repositories.begin(); it != repositories.end(); ++it) {

if ((\*it)->getName() == repoName) {

delete \*it;

repositories.erase(it);

break;

}

}

}

const vector<Repository\*>& getRepositories() const {

return repositories;

}

};

int main() {

vector<User\*> users;

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

users.push\_back(new User(username, password));

User\* loggedInUser = users.front();

cout << "Logged in as: " << loggedInUser->getUsername() << endl;

char createRepoChoice;

cout << "Create a repository? (yes/no): ";

cin >> createRepoChoice;

if (createRepoChoice == 'y' || createRepoChoice == 'Y') {

string repoName;

char visibilityChoice;

bool isPublic;

cout << "Enter repository name: ";

cin >> repoName;

cout << "Is the repository public? (yes/no): ";

cin >> visibilityChoice;

isPublic = (visibilityChoice == 'y' || visibilityChoice == 'Y');

loggedInUser->createRepository(repoName, isPublic);

cout << "Repository created: " << repoName << endl;

}

// Cleaning up memory

for (auto& user : users) {

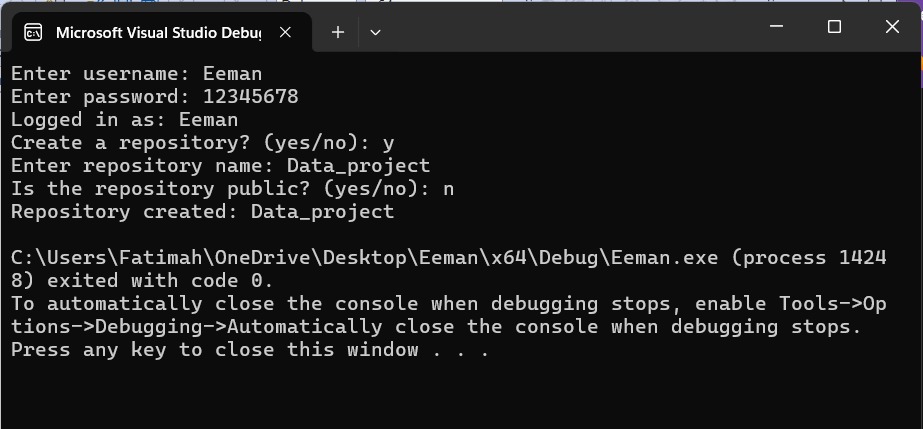
delete user;

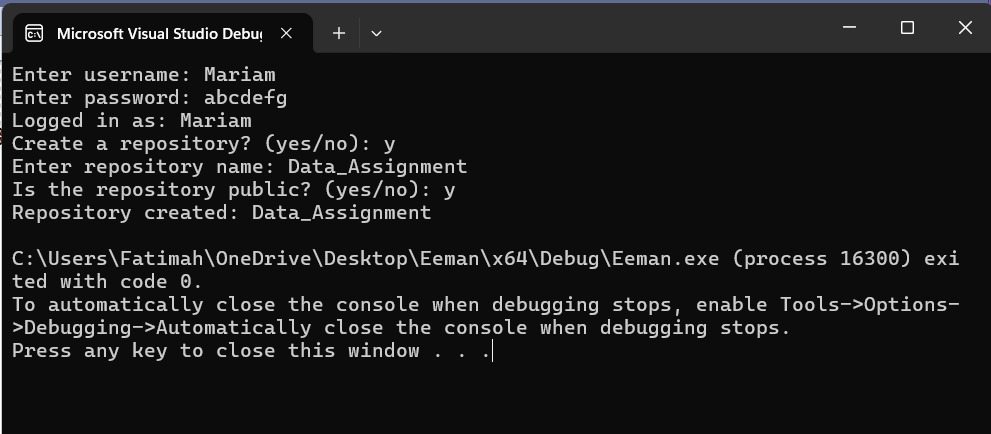
}

return 0;

}

**OUTPUT:**

****

****