**PROJECT CODE**

**FileManager.hpp:**

**// FileManager.hpp**

#ifndef FILE\_MANAGER\_HPP

#define FILE\_MANAGER\_HPP

#include <string>

class FileManager {

public:

void displayMenu();

void handleUserChoice(int choice);

private:

void moveFile();

void copyFile();

void deleteFile();

void createFile();

void searchFile();

void managePermissions();

**// Helper functions to get input from the user**

std::string getFilePath(const std::string &prompt);

std::string getSearchCriteria();

std::string getDirectory();

std::string getPermissions();

void logOperation(const std::string &operation, const std::string &details);

};

#endif // FILE\_MANAGER\_HPP

**FileManager.cpp**

#include "FileManager.hpp"

#include <iostream>

#include <filesystem>

#include <fstream>

#include <chrono>

#include <iomanip>

#include <sstream> // Include this for std::ostringstream

void FileManager::displayMenu() {

std::cout << "Available operations:\n";

std::cout << "1. Move file\n";

std::cout << "2. Copy file\n";

std::cout << "3. Delete file\n";

std::cout << "4. Create file\n";

std::cout << "5. Search file\n";

std::cout << "6. Manage file permissions\n";

std::cout << "7. Exit\n";

}

void FileManager::handleUserChoice(int choice) {

switch (choice) {

case 1: moveFile(); break;

case 2: copyFile(); break;

case 3: deleteFile(); break;

case 4: createFile(); break;

case 5: searchFile(); break;

case 6: managePermissions(); break;

case 7: std::cout << "Exit\n"; break;

default: std::cout << "Invalid choice. Please try again.\n";

}

}

void FileManager::moveFile() {

std::string source = getFilePath("Enter source file path: ");

std::string destination = getFilePath("Enter destination file path: ");

try {

std::filesystem::rename(source, destination);

std::cout << "File moved successfully.\n";

logOperation("Move", source + " -> " + destination); // Log after success

} catch (const std::filesystem::filesystem\_error &e) {

std::cerr << "Error moving file: " << e.what() << '\n';

}

}

void FileManager::copyFile() {

std::string source = getFilePath("Enter source file path: ");

std::string destination = getFilePath("Enter destination file path: ");

try {

std::filesystem::copy(source, destination);

std::cout << "File copied successfully.\n";

logOperation("Copy", source + " -> " + destination); // Log after success

} catch (const std::filesystem::filesystem\_error &e) {

std::cerr << "Error copying file: " << e.what() << '\n';

}

}

void FileManager::deleteFile() {

std::string filePath = getFilePath("Enter file path to delete: ");

try {

std::filesystem::remove(filePath);

std::cout << "File deleted successfully.\n";

logOperation("Delete", filePath); // Log after success

} catch (const std::filesystem::filesystem\_error &e) {

std::cerr << "Error deleting file: " << e.what() << '\n';

}

}

void FileManager::createFile() {

std::string filePath = getFilePath("Enter file path to create: ");

std::ofstream ofs(filePath);

if (ofs) {

ofs.close();

std::cout << "File created successfully.\n";

logOperation("Create", filePath); // Log after success

} else {

std::cerr << "Error creating file: Could not open file.\n";

}

}

void FileManager::searchFile() {

std::string criteria = getSearchCriteria();

std::string directory = getDirectory();

for (const auto &entry : std::filesystem::recursive\_directory\_iterator(directory)) {

if (entry.path().filename().string().find(criteria) != std::string::npos) {

std::cout << "Found: " << entry.path() << '\n';

}

}

logOperation("Search", criteria + " in " + directory); // Log after search

}

void FileManager::managePermissions() {

std::string filePath = getFilePath("Enter file or directory path: ");

std::string permissions = getPermissions();

try {

**// Convert string permissions to std::filesystem::perms here**

std::filesystem::permissions(filePath,

std::filesystem::perms::owner\_all |

std::filesystem::perms::group\_all |

std::filesystem::perms::others\_all);

std::cout << "Permissions modified successfully.\n";

logOperation("ManagePermissions", filePath + " -> " + permissions); // Log after success

} catch (const std::filesystem::filesystem\_error &e) {

std::cerr << "Error managing permissions: " << e.what() << '\n';

}

}

std::string FileManager::getFilePath(const std::string &prompt) {

std::string path;

std::cout << prompt;

std::cin >> path;

return path;

}

std::string FileManager::getSearchCriteria() {

std::string criteria;

std::cout << "Enter search criteria: ";

std::cin >> criteria;

return criteria;

}

std::string FileManager::getDirectory() {

std::string directory;

std::cout << "Enter directory to search in: ";

std::cin >> directory;

return directory;

}

std::string FileManager::getPermissions() {

std::string permissions;

std::cout << "Enter new permissions (e.g., rwxrwxrwx): ";

std::cin >> permissions;

return permissions;

}

void FileManager::logOperation(const std::string &operation, const std::string &details) {

**// Get the current time**

auto now = std::chrono::system\_clock::now();

std::time\_t now\_c = std::chrono::system\_clock::to\_time\_t(now);

**// Create a string stream to format the log message**

std::ostringstream oss;

oss << "[" << std::put\_time(std::localtime(&now\_c), "%Y-%m-%d %H:%M:%S") << "] " <<

operation << ": " << details << '\n';

std::string logMessage = oss.str();

**// Print log message to console**

std::cout << logMessage;

**// Open the log file**

std::ofstream logFile("file\_manager.log", std::ios::app);

if (logFile) {

**// Write the log message to the file**

logFile << logMessage;

} else {

std::cerr << "Error: Could not open log file.\n";

}

}

**main.cpp:**

**// main.cpp**

#include "FileManager.hpp"

#include <iostream>

int main() {

FileManager fm;

int choice = 0;

while (choice != 7) {

fm.displayMenu();

std::cout << "Enter your choice: ";

std::cin >> choice;

fm.handleUserChoice(choice);

}

return 0;

}

**Makefile:**

# Makefile

# Compiler

CXX = g++

# Compiler flags

CXXFLAGS = -std=c++17 -Wall -Wextra

# Target executable

TARGET = file\_manager

# Source files

SRCS = main.cpp FileManager.cpp

# Object files

OBJS = $(SRCS:.cpp=.o)

# Default rule

all: $(TARGET)

# Link object files to create the executable

$(TARGET): $(OBJS)

$(CXX) $(CXXFLAGS) -o $@ $(OBJS)

# Rule to compile .cpp files to .o files

%.o: %.cpp

$(CXX) $(CXXFLAGS) -c $< -o $@

# Clean up build artifacts

**clean:**

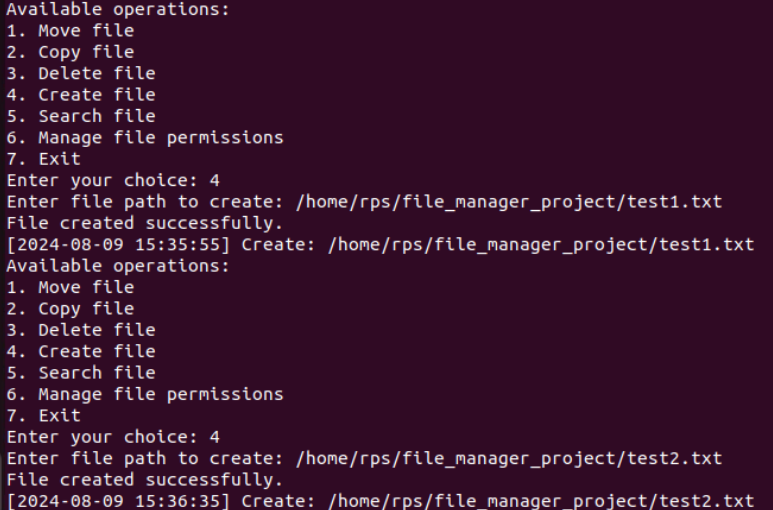
rm -f $(TARGET) $(OBJS)

# Phony targets

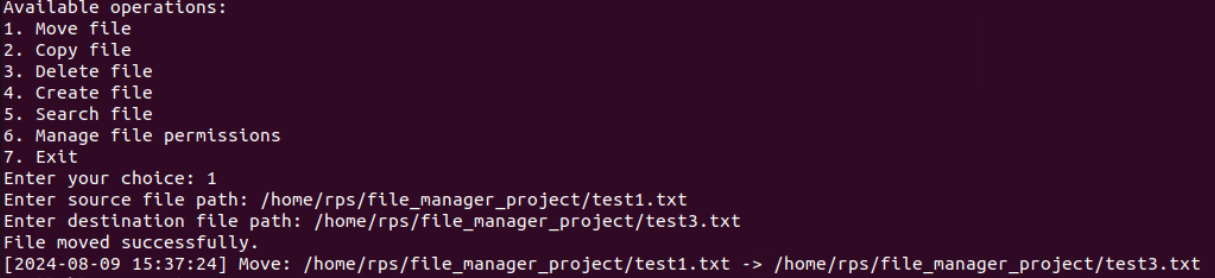
**PHONY:** all clean

**Outputs:**

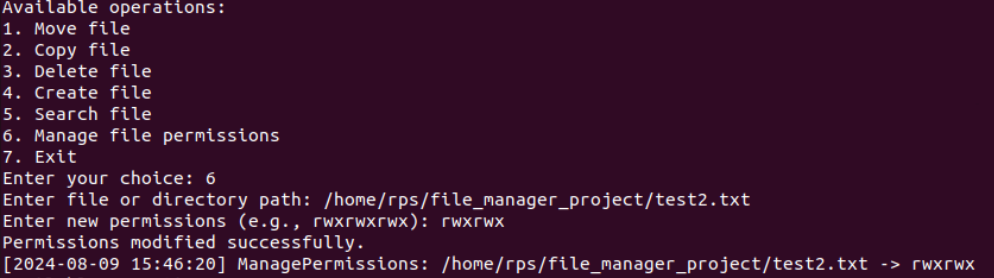
**Create:**



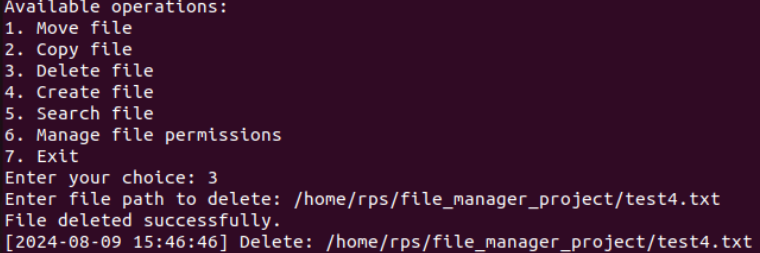
**Move:**



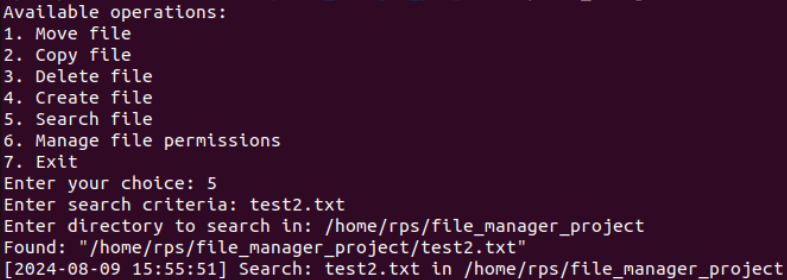
**Manage file permissions:**



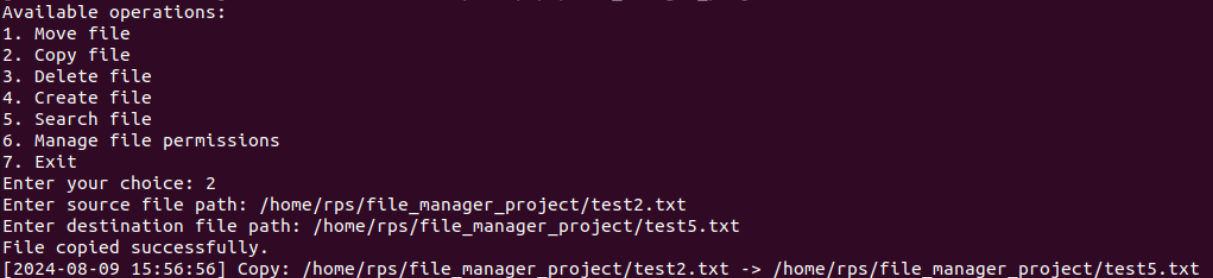
**Delete:**



**Search:**



**Copy:**

****

**Logger:**

