Simple File Explorer in C++

1.Objective

The main objective of this project is to develop a **console-based File Explorer application** in C++ that allows users to perform common file system operations such as:

- Listing files and directories
- Changing directories
- Creating files and folders
- Deleting files and folders
- Reading file contents

This helps in understanding how **file handling, directories, and system calls** work in C++ on a Unix/Linux or Windows (MinGW) environment.

2.Tools & Technologies Used

- **Language:** C++
- **Compiler:** g++ (MinGW / Linux GCC)
- **Libraries:**
    - <iostream>
    - <fstream>
    - <dirent.h>
    - <sys/stat.h>
    - <unistd.h>
    - <cstring>
    - <limits>

3. Features

✅ List all files and folders in the current directory
✅ Change directory (including going up with . .)
✅ Create new files and directories
✅ Delete existing files or folders

✅ Display the current path
✅ Read and display file content in the terminal

4. Source code

```
        #include <iostream>
#include <fstream>
#include <string>
#include <limits>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>
#include <cstring>
using namespace std;


void listFiles(const string& currentPath) {
    cout << "\nFiles and directories in: " << currentPath << endl;
    cout << "----------------------------------------" << endl;
    DIR* dir = opendir(currentPath.c_str());
    if (dir == nullptr) {
        cerr << "Error accessing directory: " << strerror(errno) << endl;
        cout << "----------------------------------------" << endl;
        return;
    }
    struct dirent* entry;
    while ((entry = readdir(dir)) != nullptr) {
        string name = entry->d_name;
```

```
        if (name == "." || name == "..") continue;
        string fullPath = currentPath + "/" + name;
        struct stat statbuf;
        if (stat(fullPath.c_str(), &statbuf) == 0) {
            if (S_ISDIR(statbuf.st_mode))
                cout << "[DIR]  " << name << endl;
            else
                cout << "       " << name << endl;
        }
    }
    closedir(dir);
    cout << "----------------------------------------" << endl;
}

void changeDirectory(string& currentPath) {
    cout << "Enter directory name (or .. to go up): ";
    string dir;
    getline(cin, dir);
    if (dir.empty()) {
        cout << "Directory name cannot be empty.\n";
        return;
    }
    string newPath = (dir == "..") ? currentPath.substr(0, currentPath.find_last_of('/')) :
currentPath + "/" + dir;
    struct stat statbuf;
    if (stat(newPath.c_str(), &statbuf) == 0 && S_ISDIR(statbuf.st_mode)) {
```

```
        if (chdir(newPath.c_str()) == 0) {
            char cwd[PATH_MAX];
            if (getcwd(cwd, sizeof(cwd)) != nullptr) {
                currentPath = cwd;
                cout << "Changed directory to: " << currentPath << endl;
            } else {
                cerr << "Error getting current directory: " << strerror(errno) << endl;
            }
        } else {
            cerr << "Error changing directory: " << strerror(errno) << endl;
        }
    } else {
        cout << "Invalid directory.\n";
    }
}

void createFile(const string& currentPath) {
    cout << "Enter new file name: ";
    string filename;
    getline(cin, filename);
    if (filename.empty()) {
        cout << "File name cannot be empty.\n";
        return;
    }
    string filePath = currentPath + "/" + filename;
```

```
    ofstream file(filePath);
    if (file) {
        cout << "File created: " << filePath << endl;
    } else {
        cerr << "Failed to create file.\n";
    }
}

void createDirectory(const string& currentPath) {
    cout << "Enter new folder name: ";
    string folder;
    getline(cin, folder);
    if (folder.empty()) {
        cout << "Folder name cannot be empty.\n";
        return;
    }
    string dirPath = currentPath + "/" + folder;
    if (mkdir(dirPath.c_str(), 0755) == 0)
        cout << "Directory created: " << dirPath << endl;
    else
        cerr << "Failed to create directory: " << strerror(errno) << endl;
}

void deleteEntry(const string& currentPath) {
    cout << "Enter file or folder name to delete: ";
```

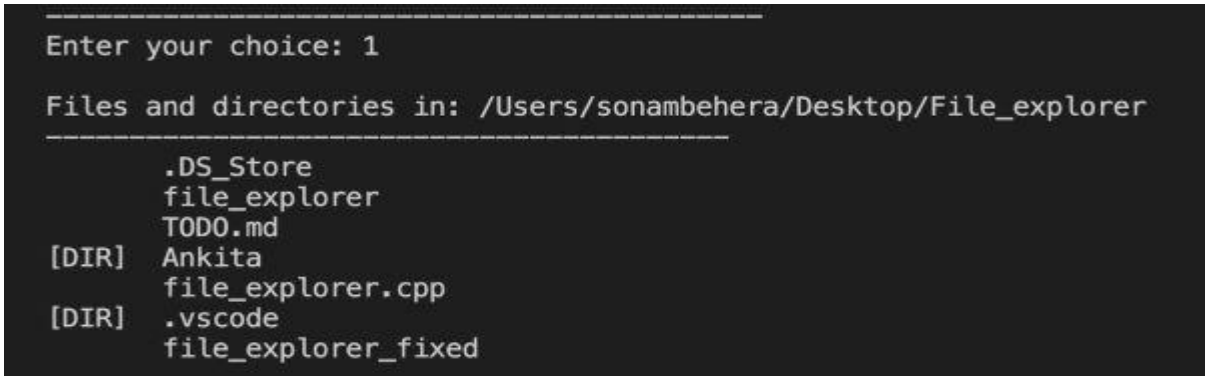
```
    string name;
    getline(cin, name);
    if (name.empty()) {
        cout << "Name cannot be empty.\n";
        return;
    }
    string target = currentPath + "/" + name;
    struct stat statbuf;
    if (stat(target.c_str(), &statbuf) != 0) {
        cout << "File or directory not found.\n";
        return;
    }
    if (S_ISDIR(statbuf.st_mode)) {
        if (rmdir(target.c_str()) == 0)
            cout << "Directory deleted: " << target << endl;
        else
            cerr << "Error deleting directory: " << strerror(errno) << endl;
    } else {
        if (remove(target.c_str()) == 0)
            cout << "File deleted: " << target << endl;
        else
            cerr << "Error deleting file: " << strerror(errno) << endl;
    }
}
```

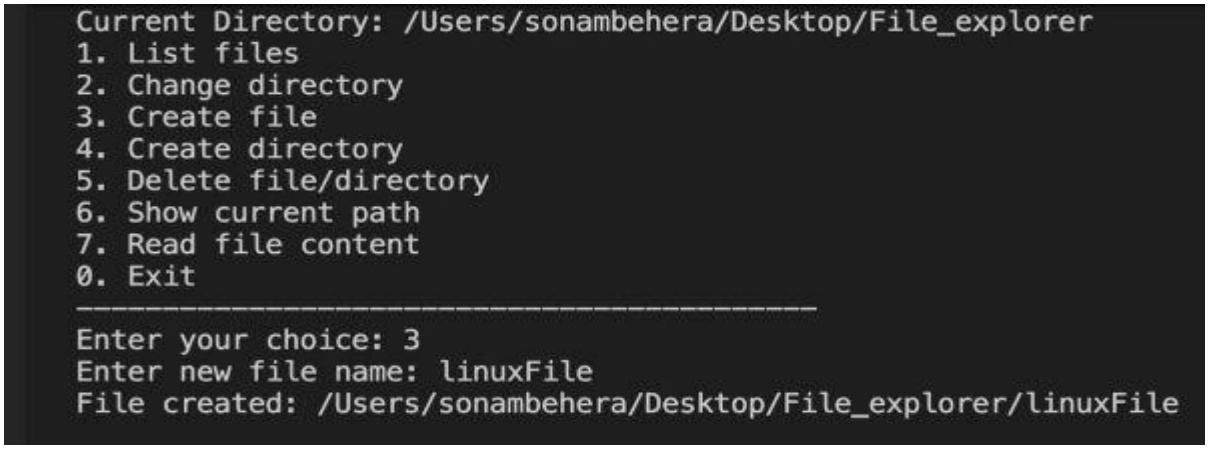
```
void readFile(const string& currentPath) {
    cout << "Enter file name to read: ";
    string filename;
    getline(cin, filename);
    if (filename.empty()) {
        cout << "File name cannot be empty.\n";
        return;
    }
    string filePath = currentPath + "/" + filename;
    struct stat statbuf;
    if (stat(filePath.c_str(), &statbuf) != 0 || S_ISDIR(statbuf.st_mode)) {
        cout << "File not found or is a directory.\n";
        return;
    }
    ifstream file(filePath);
    if (file) {
        cout << "\nContent of " << filePath << ":\n";
        cout << "----------------------------------------" << endl;
        string line;
        while (getline(file, line)) {
            cout << line << endl;
        }
        cout << "----------------------------------------" << endl;
    } else {
        cerr << "Failed to open file.\n";
```

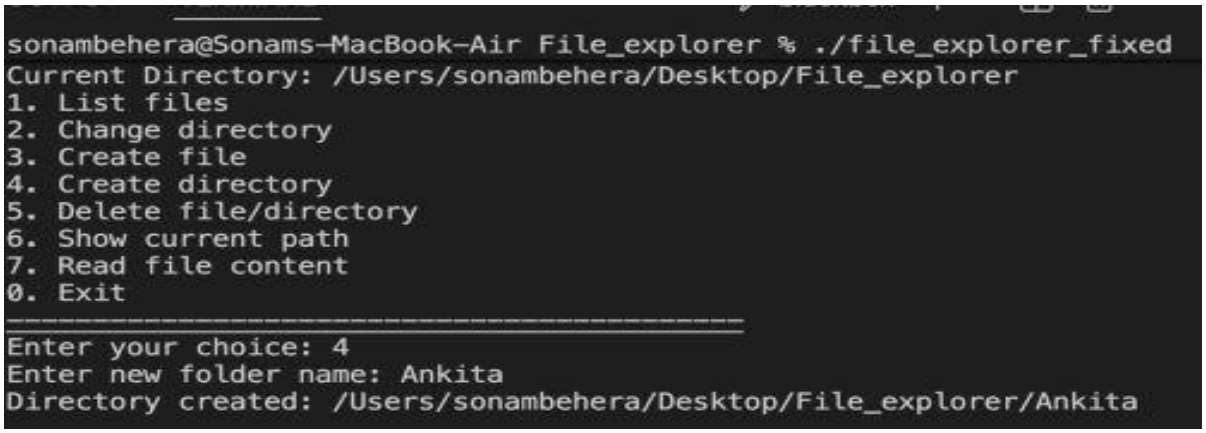
```
    }
}

int main() {
    char cwd[PATH_MAX];
    if (getcwd(cwd, sizeof(cwd)) == nullptr) {
        cerr << "Error getting current directory: " << strerror(errno) << endl;
        return 1;
    }
    string currentPath = cwd;
    int choice = -1;
    do {
        cout << "\n========== SIMPLE FILE EXPLORER ==========" << endl;
        cout << "Current Directory: " << currentPath << endl;
        cout << "1. List files" << endl;
        cout << "2. Change directory" << endl;
        cout << "3. Create file" << endl;
        cout << "4. Create directory" << endl;
        cout << "5. Delete file/directory" << endl;
        cout << "6. Show current path" << endl;
        cout << "7. Read file content" << endl;
        cout << "0. Exit" << endl;
        cout << "-----------------------------------------" << endl;
        cout << "Enter your choice: ";
        if (!(cin >> choice)) {
```

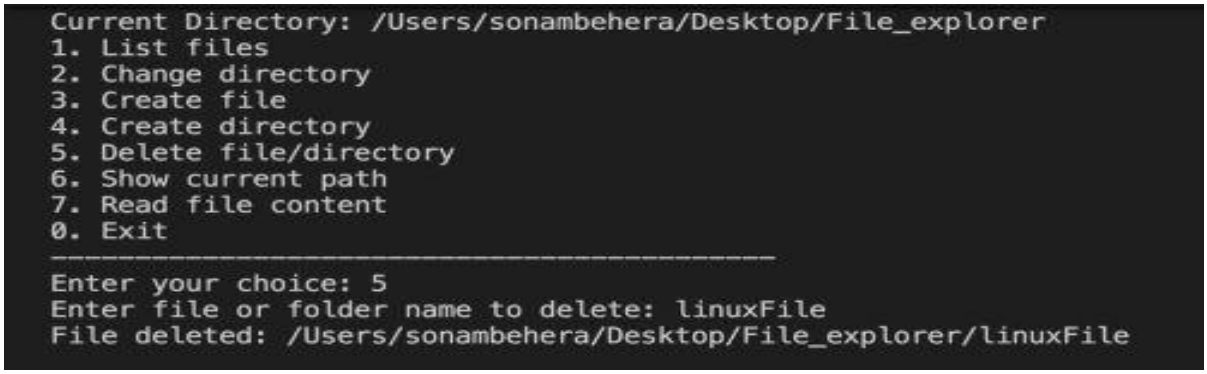
```
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Invalid input! Please enter a number.\n";
            continue;
        }
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        switch (choice) {
            case 1: listFiles(currentPath); break;
            case 2: changeDirectory(currentPath); break;
            case 3: createFile(currentPath); break;
            case 4: createDirectory(currentPath); break;
            case 5: deleteEntry(currentPath); break;
            case 6: cout << "Current Path: " << currentPath << endl; break;
            case 7: readFile(currentPath); break;
            case 0: cout << "Exiting...\n"; break;
            default: cout << "Invalid choice!\n"; break;
        }
    } while (choice != 0);
    return 0;
}
```

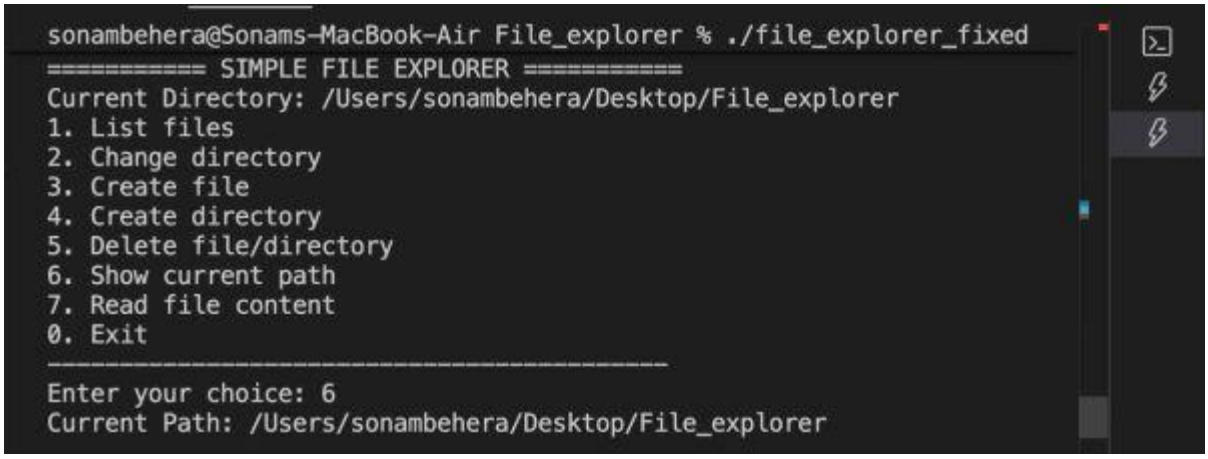
5. Screenshots to Include

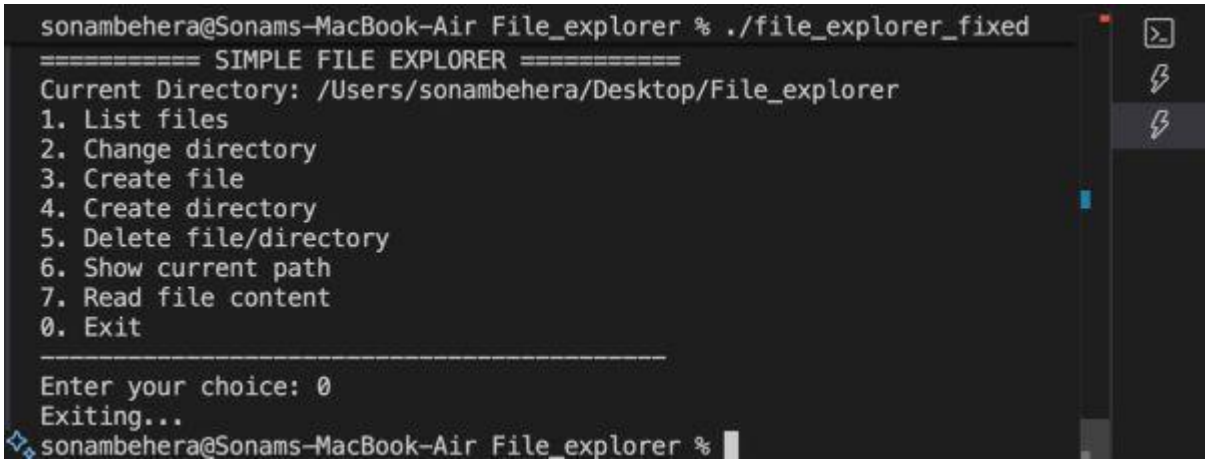












6. Output Explanation

The program provides a simple menu-driven interface that allows users to explore and manipulate files in their current directory. It uses standard C++ I/O and POSIX directory APIs to perform operations safely.

7. Conclusion

This project demonstrates the practical use of C++ file handling, directory traversal, and system calls in a real-world console application. It helps strengthen understanding of file system structures and user interaction via CLI.