File Explorer Capstone Project Report

The **File Explorer** project is a desktop application that allows users to manage files and folders efficiently. It supports creating, deleting, renaming, and viewing files and directories. The project demonstrates practical file handling and directory management while providing a user-friendly interface. It enables users to perform essential operations such as listing, creating, deleting, copying, moving, and searching files and folders. The project utilizes the modern **C++17 <filesystem>** library to handle directory traversal and file manipulation securely and efficiently.

Key Features:

1 List files and directories with size and permissions
2 Create, delete, move, and copy files/folders
3 View file content using 'cathead' and 'cattail' commands
4 Recursive search using regex patterns
5 Change file permissions using chmod-like syntax
6 Interactive help and user-friendly command-line interface

Novelty (Highlight):

■ Added advanced commands like **search** (regex-based file search) and **chmod** (permission control), which extend beyond the standard basic file operations .

Code File:

The complete source code is available on Github at the following link:

- https://github.com/FarheenNaz786/File_Explorer_Project.git

```cpp
#ifdef _WIN32
#undef UNICODE
#endif

#include <iostream>
#include <string>
#include <vector>
#include <filesystem>
#include <fstream>
#include <deque>
#include <regex>
#include <sstream>
#include <system_error>
#include <iomanip>
namespace fs = std::filesystem;

struct FileInfo {
    std::string name;
    std::string path;
    bool is_dir = false;
    uintmax_t size = 0;
    std::string perms;
};

std::string perms_to_string(fs::perms p) {
    // Show rwx for owner/group/others (9 chars)
    std::string s = "---------";
    s[0] = (p & fs::perms::owner_read)  != fs::perms::none ? 'r' : '-';
    s[1] = (p & fs::perms::owner_write) != fs::perms::none ? 'w' : '-';
```

```
    s[2] = (p & fs::perms::owner_exec)  != fs::perms::none ? 'x' : '-';
    s[3] = (p & fs::perms::group_read)  != fs::perms::none ? 'r' : '-';
    s[4] = (p & fs::perms::group_write) != fs::perms::none ? 'w' : '-';
    s[5] = (p & fs::perms::group_exec)  != fs::perms::none ? 'x' : '-';
    s[6] = (p & fs::perms::others_read) != fs::perms::none ? 'r' : '-';
    s[7] = (p & fs::perms::others_write)!= fs::perms::none ? 'w' : '-';
    s[8] = (p & fs::perms::others_exec) != fs::perms::none ? 'x' : '-';
    return s;
}

std::vector<FileInfo> list_directory(const std::string &path) {
    std::vector<FileInfo> out;
    std::error_code ec;
    fs::directory_iterator it(path, ec);
    if (ec) {
        std::cerr << "ls: cannot access '" << path << "': " << ec.message() << '\n';
        return out;
    }
    for (const auto &entry : it) {
        FileInfo fi;
        fi.name = entry.path().filename().string();
        fi.path = entry.path().string();
        std::error_code ec2;
        auto st = entry.symlink_status(ec2);
        if (!ec2) {
            fi.is_dir = fs::is_directory(st);
            try {
                if (fs::is_regular_file(st)) {
                    fi.size = fs::file_size(entry.path(), ec2);
                    if (ec2) fi.size = 0;
                } else {
                    fi.size = 0;
                }
                fi.perms = perms_to_string(st.permissions());
            } catch (...) {
                fi.size = 0;
                fi.perms = "?????????";
            }
        } else {
            fi.is_dir = false;
            fi.size = 0;
            fi.perms = "?????????";
        }
        out.push_back(fi);
    }
    return out;
}

void print_listing(const std::vector<FileInfo>& entries) {
    std::cout << std::left << std::setw(30) << "Name"
              << std::setw(8) << "Type"
              << std::setw(12) << "Size"
              << "Perms\n";
    for (auto &e : entries) {
        std::cout << std::left << std::setw(30) << e.name
                  << std::setw(8) << (e.is_dir ? "DIR" : "FILE")
                  << std::setw(12) << e.size
                  << e.perms << "\n";
    }
}

bool change_directory(const std::string &path) {
    std::error_code ec;
    fs::current_path(path, ec);
    if (ec) {
        std::cerr << "cd: " << ec.message() << '\n';
        return false;
    }
    return true;
}

std::string get_current_path() {
    std::error_code ec;
    auto p = fs::current_path(ec);
```

```
    if (ec) return std::string{};
    return p.string();
}

bool create_file(const std::string &path) {
    std::ofstream ofs(path, std::ios::out | std::ios::app);
    if (!ofs) {
        std::cerr << "mkfile: failed to create " << path << '\n';
        return false;
    }
    return true;
}

bool create_directory(const std::string &path) {
    std::error_code ec;
    bool ok = fs::create_directories(path, ec);
    if (ec) {
        std::cerr << "mkdir: " << ec.message() << '\n';
        return false;
    }
    return ok || fs::exists(path);
}

bool remove_path(const std::string &path) {
    std::error_code ec;
    uintmax_t removed = fs::remove_all(path, ec);
    if (ec) {
        std::cerr << "rm: " << ec.message() << '\n';
        return false;
    }
    return removed > 0;
}

bool copy_path(const std::string &src, const std::string &dst) {
    std::error_code ec;
    fs::copy(src, dst, fs::copy_options::recursive | fs::copy_options::overwrite_existing, ec);
    if (ec) {
        std::cerr << "cp: " << ec.message() << '\n';
        return false;
    }
    return true;
}

bool move_path(const std::string &src, const std::string &dst) {
    std::error_code ec;
    fs::rename(src, dst, ec);
    if (!ec) return true;

    // If rename fails (different FS or permission), try copy+remove
    if (!copy_path(src, dst)) return false;
    return remove_path(src);
}

bool show_file_head(const std::string &path, int lines) {
    std::ifstream ifs(path);
    if (!ifs) {
        std::cerr << "cathead: cannot open " << path << '\n';
        return false;
    }
    std::string line;
    int count = 0;
    while (count < lines && std::getline(ifs, line)) {
        std::cout << line << '\n'; ++count;
    }
    return true;
}

bool show_file_tail(const std::string &path, int lines) {
    std::ifstream ifs(path);
    if (!ifs) {
        std::cerr << "cattail: cannot open " << path << '\n';
        return false;
    }
    std::deque<std::string> buffer;
```

```
    std::string line;
    while (std::getline(ifs, line)) {
        buffer.push_back(line);
        if ((int)buffer.size() > lines) buffer.pop_front();
    }
    for (auto &l : buffer) std::cout << l << '\n';
    return true;
}

std::vector<std::string> search_recursive(const std::string &root, const std::string &pattern) {
    std::vector<std::string> results;
    std::regex re;
    try {
        re = std::regex(pattern, std::regex::ECMAScript | std::regex::icase);
    } catch (const std::exception &e) {
        std::cerr << "search: invalid regex: " << e.what() << '\n';
        return results;
    }
    std::error_code ec;
    for (auto it = fs::recursive_directory_iterator(root, ec); !ec && it != fs::recursive_directory_iterator();
it.increment(ec)) {
        if (ec) break;
        try {
            if (std::regex_search(it->path().filename().string(), re)) {
                results.push_back(it->path().string());
            }
        } catch (...) {}
    }
    if (ec) std::cerr << "search: " << ec.message() << '\n';
    return results;
}

// Convert string like "755" (octal) to perms and apply (owner/group/others rwx).
bool change_permissions(const std::string &path, const std::string &mode) {
    if (mode.size() != 3) {
        std::cerr << "chmod: mode should be 3 digits like 755\n";
        return false;
    }
    int m = 0;
    try {
        m = std::stoi(mode, nullptr, 8); // parse as octal
    } catch (...) {
        std::cerr << "chmod: invalid mode\n";
        return false;
    }

    fs::perms p = fs::perms::none;
    // owner
    if (m & 0400) p |= fs::perms::owner_read;
    if (m & 0200) p |= fs::perms::owner_write;
    if (m & 0100) p |= fs::perms::owner_exec;
    // group
    if (m & 0040) p |= fs::perms::group_read;
    if (m & 0020) p |= fs::perms::group_write;
    if (m & 0010) p |= fs::perms::group_exec;
    // others
    if (m & 0004) p |= fs::perms::others_read;
    if (m & 0002) p |= fs::perms::others_write;
    if (m & 0001) p |= fs::perms::others_exec;

    std::error_code ec;
    fs::permissions(path, p, ec);
    if (ec) {
        std::cerr << "chmod: " << ec.message() << '\n';
        return false;
    }
    return true;
}

void show_menu() {
    std::cout << "\n===== File Explorer =====\n";
    std::cout << "help | pwd | ls [path] | cd <p> | mkfile <p> | mkdir <p> | rm <p>\n";
    std::cout << "cp <a> <b> | mv <a> <b> | cathead <f> N | cattail <f> N\n";
    std::cout << "search <root> <regex> | chmod <p> <mode> | exit\n";
```

```cpp
}

int main() {
    std::string line;
    std::cout << "Type 'help' for menu.\n";
    while (true) {
        std::cout << "> ";
        if (!std::getline(std::cin, line)) break;
        if (line.empty()) continue;

        std::istringstream iss(line);
        std::string cmd;
        iss >> cmd;

        if (cmd == "help") { show_menu(); }
        else if (cmd == "pwd") { std::cout << get_current_path() << '\n'; }
        else if (cmd == "ls") {
            std::string p;
            if (iss >> std::ws && std::getline(iss, p) && !p.empty()) {
                // trim
                if (p.front() == ' ') p.erase(0, p.find_first_not_of(' '));
                print_listing(list_directory(p));
            } else {
                print_listing(list_directory(get_current_path()));
            }
        }
        else if (cmd == "cd") {
            std::string p; if (iss >> p) change_directory(p);
        }
        else if (cmd == "mkfile") {
            std::string p; if (iss >> p) create_file(p);
        }
        else if (cmd == "mkdir") {
            std::string p; if (iss >> p) create_directory(p);
        }
        else if (cmd == "rm") {
            std::string p; if (iss >> p) remove_path(p);
        }
        else if (cmd == "cp") {
            std::string a, b; if (iss >> a >> b) copy_path(a, b);
        }
        else if (cmd == "mv") {
            std::string a, b; if (iss >> a >> b) move_path(a, b);
        }
        else if (cmd == "cathead") {
            std::string f; int n; if (iss >> f >> n) show_file_head(f, n);
        }
        else if (cmd == "cattail") {
            std::string f; int n; if (iss >> f >> n) show_file_tail(f, n);
        }
        else if (cmd == "search") {
            std::string r, p;
            if (iss >> r >> p) {
                for (auto &x : search_recursive(r, p)) std::cout << x << '\n';
            } else {
                std::cerr << "search: usage: search <root> <regex>\n";
            }
        }
        else if (cmd == "chmod") {
            std::string p, m; if (iss >> p >> m) change_permissions(p, m);
        }
        else if (cmd == "exit") break;
        else {
            std::cerr << "Unknown command: " << cmd << " (type 'help')\n";
        }
    }
    return 0;
}
```

Screenshots:

```
91993@Farheen MINGW64 ~
$ cd "c:/Users/91993/Desktop/File_explorer_application"

91993@Farheen MINGW64 /c/Users/91993/Desktop/File_explorer_application
$ g++ -std=c++17 -Wall -O2 file_explorer.cpp -o file_explorer.exe

91993@Farheen MINGW64 /c/Users/91993/Desktop/File_explorer_application
$ ./file_explorer.exe
Type 'help' for menu.
> help

===== File Explorer =====
help | pwd | ls [path] | cd <p> | mkfile <p> | mkdir <p> | rm <p>
cp <a> <b> | mv <a> <b> | cathead <f> N | cattail <f> N
search <root> <regex> | chmod <p> <mode> | exit
> pwd
C:\Users\91993\Desktop\File_explorer_application
> ls
Name                          Type    Size        Perms
.vscode                       DIR     0           rwxrwxrwx
file_explorer.cpp             FILE    10179       rw-rw-rw-
file_explorer.exe             FILE    396961      rwxrwxrwx
> mkdir demo_folder
> mkdir demo_folder2
> mkfile demo_folder/hello.txt
> mkfile demo_folder2/world.txt
> ls demo_folder
Name                          Type    Size        Perms
hello.txt                     FILE    0           rw-rw-rw-
> ls demo_folder2
Name                          Type    Size        Perms
world.txt                     FILE    0           rw-rw-rw-
> mv demo_folder/hello.txt demo_folder2/hello.txt
> cp demo_folder2/hello.txt demo_folder/hello.txt
> rm demo_folder2/hello.txt
> rm demo_folder2
> cd demo_folder
> pwd
C:\Users\91993\Desktop\File_explorer_application\demo_folder
> cd ..
> cathead demo_folder/hello.txt 5
> cattail demo_folder/hello.txt 5
> search . hello
.\demo_folder\hello.txt
> chmod demo_folder/hello.txt 644
> exit

91993@Farheen MINGW64 /c/Users/91993/Desktop/File_explorer_application
$ ls
demo_folder  file_explorer.cpp  file_explorer.exe

91993@Farheen MINGW64 /c/Users/91993/Desktop/File_explorer_application
$ ls demo_folder
hello.txt

91993@Farheen MINGW64 /c/Users/91993/Desktop/File_explorer_application
$
```

Conclusion:

The File Explorer Capstone Project demonstrates the use of modern C++ features to build a lightweight, interactive command-line utility for file system operations. It showcases fundamental system-level programming skills and understanding of C++17's filesystem library, making it an ideal project for showcasing technical proficiency during interviews