Capstone Project Report

Assignment 1 (Linux OS)
Project Title: File Explorer Application in C++17
Author: Prakash Shaw **Regno**.: 2241019425
Course: Capstone Project **Batch no.:** 09
Year: 2025

Objective
Develop a console-based file explorer application in C++ that interfaces with the Linux operating system to manage files and directories. The program allows users to list files, navigate directories, manipulate files (copy, move, delete, create), perform recursive search, and manage file permissions — all through command-line commands.

Day-wise Tasks
Day 1: Designed the application structure and set up the development environment. Implemented basic file operations such as listing files in a directory (ls, pwd).
Day 2: Implemented file and directory navigation features allowing users to move through directories (cd, pwd).
Day 3: Added file manipulation capabilities — copy, move, delete, and create files (cp, mv, rm, touch, mkdir).
Day 4: Implemented recursive file search functionality (search).
Day 5: Added file permission management features (chmod).

Source Code

```
#include <filesystem>
#include <iostream>
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>
#include <algorithm>
#include <system_error>
#include <fstream>

namespace fs = std::filesystem;

// Task 1
std::string perms_to_string(fs::perms p) {
    std::string s;
    s += ((p & fs::perms::owner_read) != fs::perms::none) ? 'r' : '-';
    s += ((p & fs::perms::owner_write) != fs::perms::none) ? 'w' : '-';
    s += ((p & fs::perms::owner_exec) != fs::perms::none) ? 'x' : '-';
    s += ((p & fs::perms::group_read) != fs::perms::none) ? 'r' : '-';
    s += ((p & fs::perms::group_write) != fs::perms::none) ? 'w' : '-';
```

```
    s += ((p & fs::perms::group_exec) != fs::perms::none) ? 'x' : '-';
    s += ((p & fs::perms::others_read) != fs::perms::none) ? 'r' : '-';
    s += ((p & fs::perms::others_write) != fs::perms::none) ? 'w' : '-';
    s += ((p & fs::perms::others_exec) != fs::perms::none) ? 'x' : '-';
    return s;
}

// Task 2
void cmd_ls(const fs::path &p) {
    std::error_code ec;
    if (!fs::exists(p, ec)) {
        std::cout << "Path does not exist: " << p << "\n";
        return;
    }
    if (!fs::is_directory(p, ec)) {
        std::cout << p << " is not a directory\n";
        return;
    }
    std::cout << std::left << std::setw(12) << "Permissions"
              << std::setw(10) << "Size"
              << std::setw(12) << "Type"
              << "Name\n";
    for (auto &entry : fs::directory_iterator(p, ec)) {
        if (ec) break;
        fs::file_status st = entry.symlink_status(ec);
        std::string type = "file";
        if (fs::is_directory(st)) type = "dir";
        else if (fs::is_symlink(st)) type = "symlink";
        uintmax_t size = 0;
        if (fs::is_regular_file(st)) size = fs::file_size(entry.path(), ec);
        std::string perms = perms_to_string(st.permissions());
        std::cout << std::left << std::setw(12) << perms
                  << std::setw(10) << size
                  << std::setw(12) << type
                  << entry.path().filename().string()
                  << "\n";
    }
}

// Task 3
void cmd_pwd(const fs::path &cwd) {
    std::cout << cwd << "\n";
}

void cmd_cd(fs::path &cwd, const std::string &arg) {
```

```
    fs::path newp;
    if (arg == "..") newp = cwd.parent_path();
    else if (arg == "-") {
        std::cout << "Use absolute or relative path. '-' not supported here.\n";
        return;
    } else newp = cwd / arg;
    std::error_code ec;
    fs::path resolved = fs::weakly_canonical(newp, ec);
    if (ec || !fs::exists(resolved, ec) || !fs::is_directory(resolved, ec)) {
        std::cout << "Cannot change directory to: " << newp << "\n";
        return;
    }
    cwd = resolved;
}

// Task 4
void show_file(const fs::path &p) {
    std::error_code ec;
    if (!fs::exists(p, ec) || !fs::is_regular_file(p, ec)) {
        std::cout << "File does not exist or not a regular file: " << p << "\n";
        return;
    }
    std::ifstream in(p);
    std::string line;
    while (std::getline(in, line)) {
        std::cout << line << "\n";
    }
}

bool copy_with_overwrite(const fs::path &src, const fs::path &dst) {
    std::error_code ec;
    fs::copy_file(src, dst, fs::copy_options::overwrite_existing, ec);
    if (ec) {
        std::cerr << "Copy failed: " << ec.message() << "\n";
        return false;
    }
    return true;
}

bool move_file(const fs::path &src, const fs::path &dst) {
    std::error_code ec;
    fs::rename(src, dst, ec);
    if (!ec) return true;
    if (fs::is_directory(dst.parent_path(), ec) || !ec) {
        if (fs::is_directory(src, ec)) {
```

```
            std::cerr << "Move directory fallback not implemented.\n";
            return false;
        } else {
            if (copy_with_overwrite(src, dst)) {
                fs::remove(src, ec);
                if (ec) std::cerr << "Warning: removed source failed: " << ec.message() << "\n";
                return true;
            }
        }
    }
    return false;
}

void remove_path(const fs::path &p) {
    std::error_code ec;
    if (!fs::exists(p, ec)) { std::cout << "Path doesn't exist\n"; return; }
    if (fs::is_directory(p, ec)) {
        std::cout << "Removing directory recursively: " << p << "\n";
        fs::remove_all(p, ec);
        if (ec) std::cout << "Remove failed: " << ec.message() << "\n";
        else std::cout << "Removed\n";
    } else {
        fs::remove(p, ec);
        if (ec) std::cout << "Remove failed: " << ec.message() << "\n";
        else std::cout << "Removed\n";
    }
}

void search_recursive(const fs::path &start, const std::string &pattern) {
    std::error_code ec;
    if (!fs::exists(start, ec)) { std::cout << "Start path doesn't exist\n"; return; }
    for (auto it = fs::recursive_directory_iterator(start, ec); it != fs::recursive_directory_iterator(); ++it) {
        if (ec) break;
        try {
            if (it->path().filename().string().find(pattern) != std::string::npos) {
                std::cout << it->path() << "\n";
            }
        } catch (...) { }
    }
    if (ec) std::cout << "Search stopped: " << ec.message() << "\n";
}

fs::perms octal_to_perms(int mode) {
    fs::perms p = fs::perms::none;
    int owner = (mode >> 6) & 7;
```

```
    int group = (mode >> 3) & 7;
    int others = mode & 7;
    if (owner & 4) p |= fs::perms::owner_read;
    if (owner & 2) p |= fs::perms::owner_write;
    if (owner & 1) p |= fs::perms::owner_exec;
    if (group & 4) p |= fs::perms::group_read;
    if (group & 2) p |= fs::perms::group_write;
    if (group & 1) p |= fs::perms::group_exec;
    if (others & 4) p |= fs::perms::others_read;
    if (others & 2) p |= fs::perms::others_write;
    if (others & 1) p |= fs::perms::others_exec;
    return p;
}

void cmd_chmod(const fs::path &p, const std::string &mode_str) {
    std::error_code ec;
    if (!fs::exists(p, ec)) { std::cout << "Path doesn't exist\n"; return; }
    int mode = 0;
    try {
        mode = std::stoi(mode_str, nullptr, 8);
    } catch (...) {
        std::cout << "Invalid mode. Provide octal like 755 or 0755\n"; return;
    }
    fs::perms newp = octal_to_perms(mode);
    fs::permissions(p, newp, ec);
    if (ec) std::cout << "chmod failed: " << ec.message() << "\n";
    else std::cout << "Permissions updated\n";
}

// Task 5
std::vector<std::string> split_args(const std::string &line) {
    std::istringstream iss(line);
    std::vector<std::string> parts;
    std::string w;
    while (iss >> std::quoted(w)) {
        parts.push_back(w);
    }
    if (parts.empty()) {
        std::istringstream iss2(line);
        while (iss2 >> w) parts.push_back(w);
    }
    return parts;
}

void print_help() {
```

```
    std::cout << "Commands:\n"
              << " ls [path]           - list files (default cwd)\n"
              << " pwd                 - print current directory\n"
              << " cd <dir>            - change directory (relative or absolute)\n"
              << " cat <file>          - show file contents\n"
              << " cp <src> <dst>      - copy file\n"
              << " mv <src> <dst>      - move/rename file\n"
              << " rm <path>           - remove file or directory (recursive for dir)\n"
              << " touch <file>        - create empty file\n"
              << " mkdir <dir>         - create directory\n"
              << " search <pattern> [start-path] - recursive search for pattern in filenames\n"
              << " chmod <octal> <path>  - change permissions (e.g. 755)\n"
              << " help                - show this help\n"
              << " exit                - quit\n";
}

int main() {
    fs::path cwd = fs::current_path();
    std::string line;
    std::cout << "Simple File Explorer (C++17) - type 'help' for commands\n";
    while (true) {
        std::cout << cwd.string() << " $ ";
        if (!std::getline(std::cin, line)) break;
        if (line.empty()) continue;
        auto parts = split_args(line);
        if (parts.empty()) continue;
        const std::string cmd = parts[0];
        if (cmd == "exit") break;
        else if (cmd == "help") print_help();
        else if (cmd == "ls") {
            if (parts.size() == 1) cmd_ls(cwd);
            else cmd_ls(fs::path(parts[1]));
        } else if (cmd == "pwd") cmd_pwd(cwd);
        else if (cmd == "cd") {
            if (parts.size() < 2) { std::cout << "cd needs a directory\n"; }
            else cmd_cd(cwd, parts[1]);
        } else if (cmd == "cat") {
            if (parts.size() < 2) { std::cout << "cat needs a filename\n"; }
            else show_file(cwd / parts[1]);
        } else if (cmd == "cp") {
            if (parts.size() < 3) { std::cout << "Usage: cp src dst\n"; }
            else {
                fs::path src = cwd / parts[1];
                fs::path dst = cwd / parts[2];
                copy_with_overwrite(src, dst);
```

```
        }
    } else if (cmd == "mv") {
        if (parts.size() < 3) { std::cout << "Usage: mv src dst\n"; }
        else {
            fs::path src = cwd / parts[1];
            fs::path dst = cwd / parts[2];
            if (move_file(src, dst)) std::cout << "Moved\n";
            else std::cout << "Move failed\n";
        }
    } else if (cmd == "rm") {
        if (parts.size() < 2) { std::cout << "Usage: rm path\n"; }
        else remove_path(cwd / parts[1]);
    } else if (cmd == "touch") {
        if (parts.size() < 2) { std::cout << "Usage: touch filename\n"; }
        else {
            fs::path f = cwd / parts[1];
            std::ofstream out(f, std::ios::app);
            out.close();
            std::cout << "Touched " << f << "\n";
        }
    } else if (cmd == "mkdir") {
        if (parts.size() < 2) { std::cout << "Usage: mkdir dirname\n"; }
        else {
            fs::path d = cwd / parts[1];
            std::error_code ec;
            if (fs::create_directories(d, ec)) std::cout << "Created " << d << "\n";
            else std::cout << "Could not create: " << ec.message() << "\n";
        }
    } else if (cmd == "search") {
        if (parts.size() < 2) { std::cout << "Usage: search pattern [start-path]\n"; }
        else {
            fs::path start = (parts.size() >= 3) ? fs::path(parts[2]) : cwd;
            search_recursive(start, parts[1]);
        }
    } else if (cmd == "chmod") {
        if (parts.size() < 3) { std::cout << "Usage: chmod octal path\n"; }
        else cmd_chmod(cwd / parts[2], parts[1]);
    } else {
        std::cout << "Unknown command: " << cmd << " (type help)\n";
    }
  }
  std::cout << "Bye\n";
  return 0;
}
```

Build Command:

g++ -std=c++17 file_explorer.cpp -o file_explorer

Run Command:

./file_explorer

Screenshots

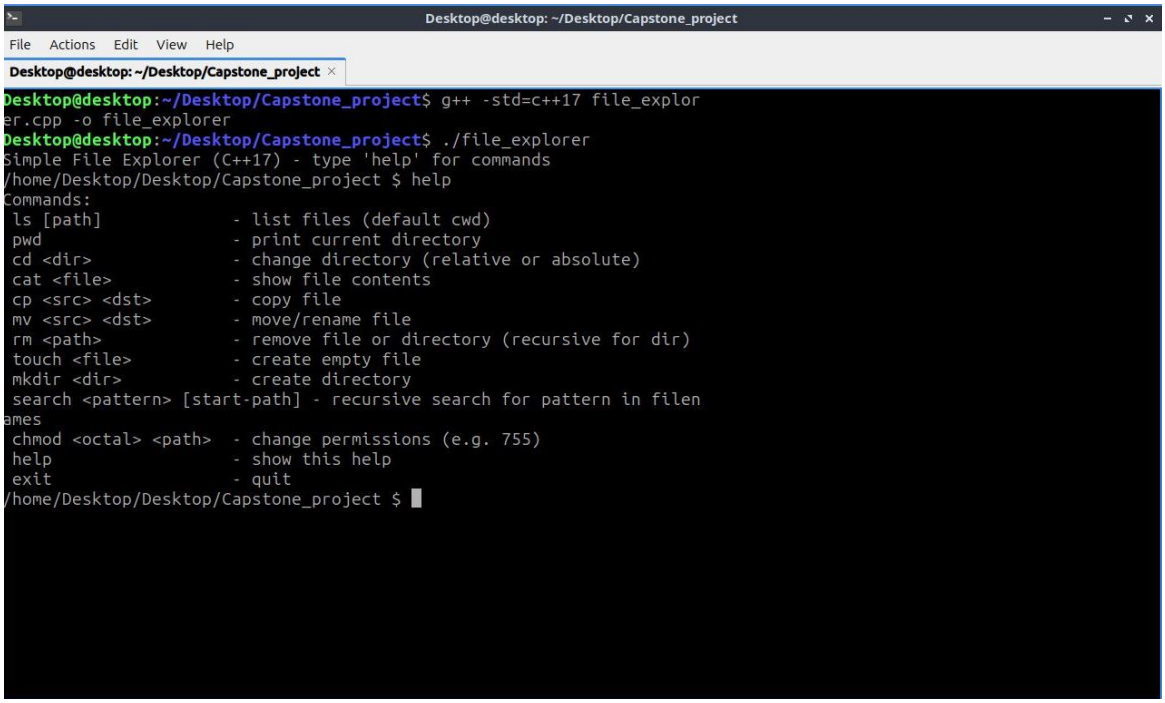

Figure 1: Output of help command

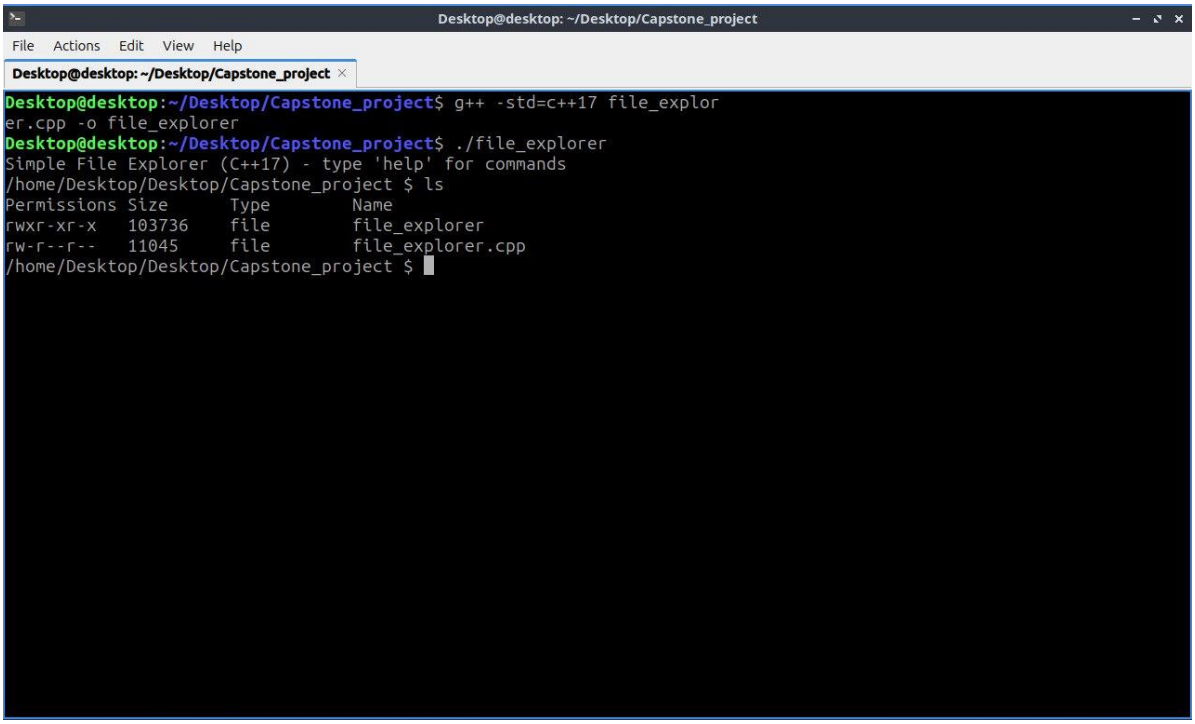

Figure 2: Output of ls command

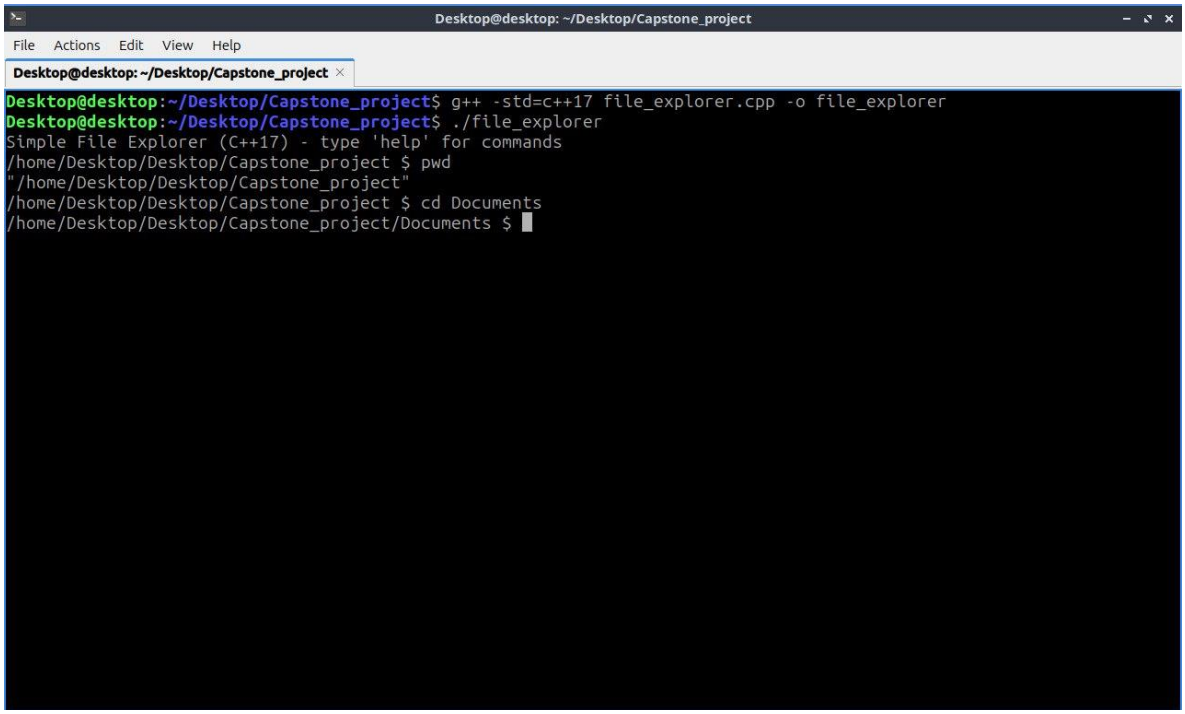

Figure 3: Output of cd and pwd commands

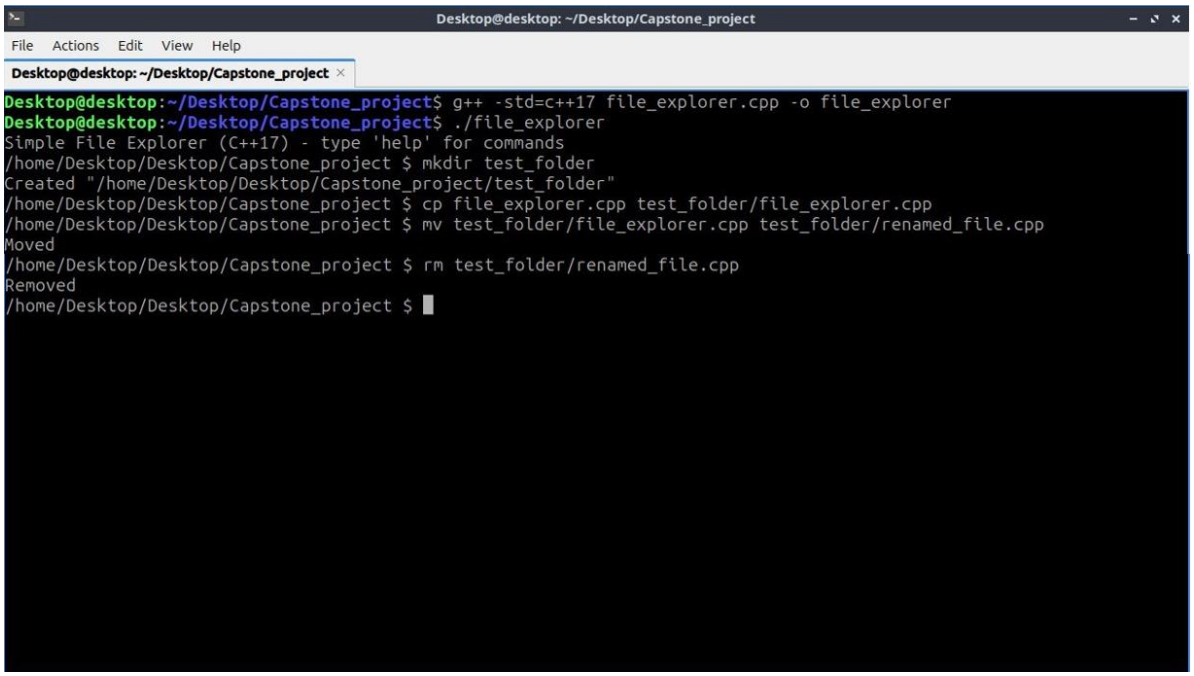

Figure 4: Output of cp, mv, rm, and mkdir commands

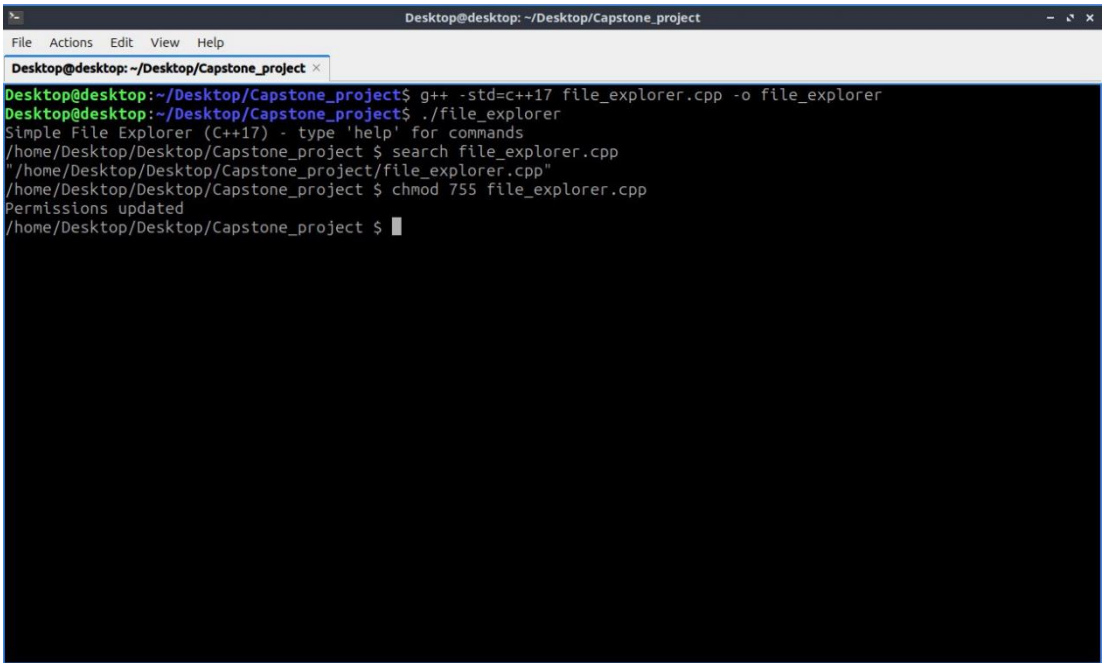

Figure 5: Output of search and chmod commands

Conclusion

The File Explorer Application successfully implements Linux-based file management through a console interface using C++17 filesystem library. It demonstrates practical use of system calls, directory traversal, and file manipulation in Linux.