INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH

(SOA Deemed to be University)

# PROJECT REPORT ON File Explorer Application (LINUX OS)

**Submitted By:**

Name: Prince Kumar Singh

Registration No.: 2241019501

Branch: CSE

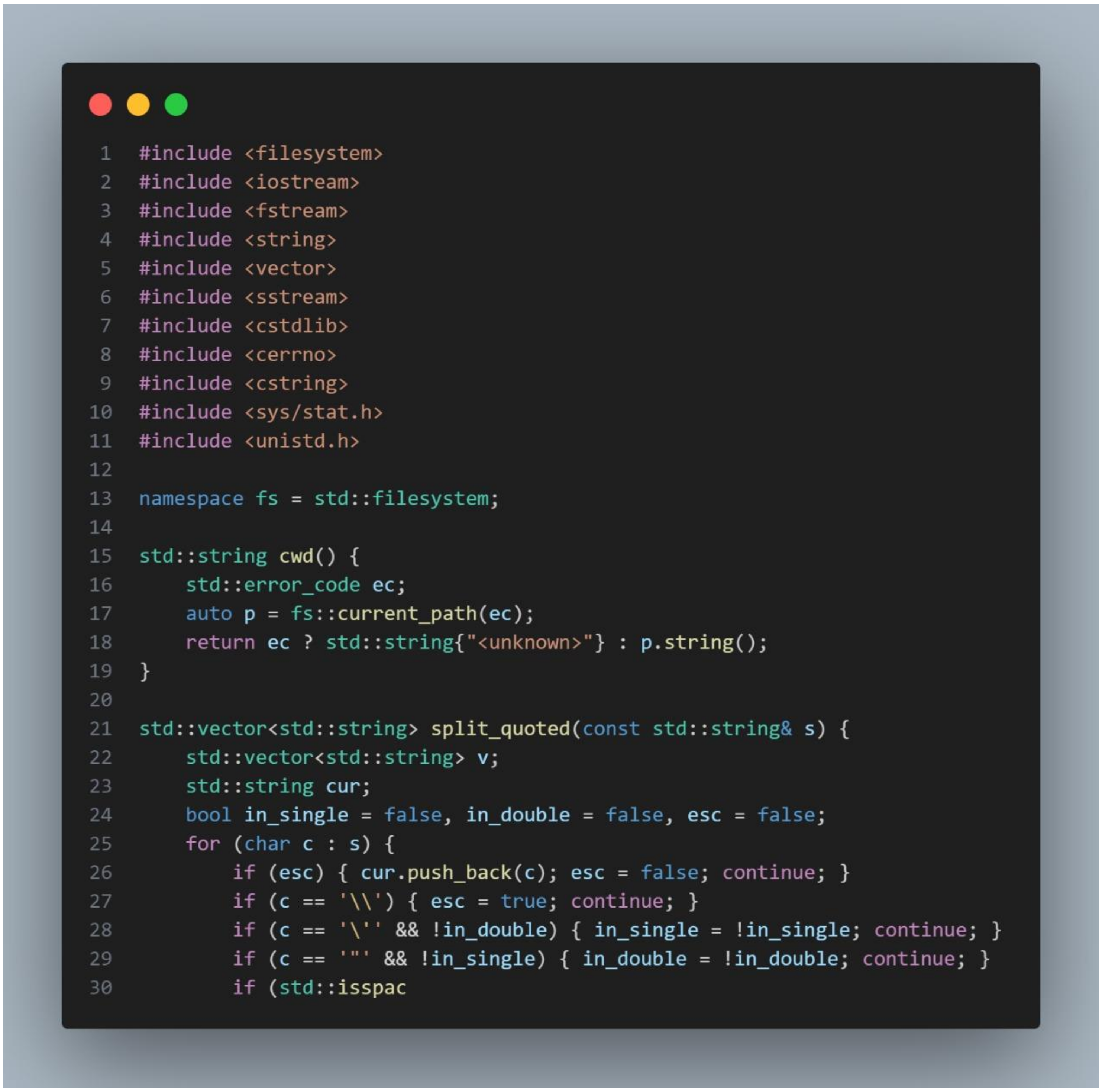
Batch: 2

## CODE:

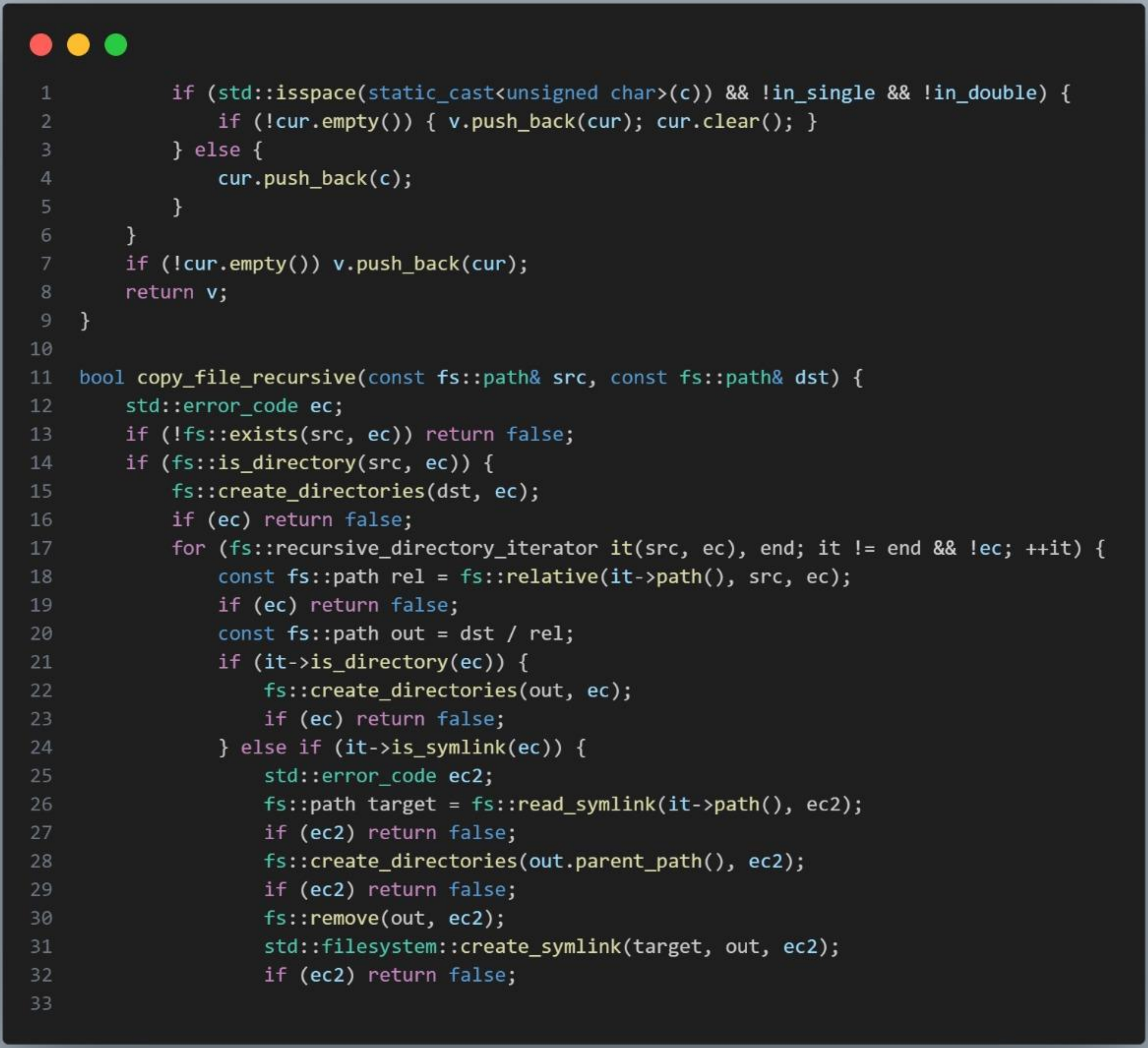
```
#include <filesystem>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <cstdlib>
#include <cerrno>
#include <cstring>
#include <sys/stat.h>
#include <unistd.h>

namespace fs = std::filesystem;

std::string cwd() {
    std::error_code ec;
    auto p = fs::current_path(ec);
    return ec ? std::string{"<unknown>"} : p.string();
}

std::vector<std::string> split_quoted(const std::string& s) {
    std::vector<std::string> v;
    std::string cur;
    bool in_single = false, in_double = false, esc = false;
    for (char c : s) {
        if (esc) { cur.push_back(c); esc = false; continue; }
        if (c == '\\') { esc = true; continue; }
        if (c == '\'' && !in_double) { in_single = !in_single; continue; }
        if (c == '"' && !in_single) { in_double = !in_double; continue; }
        if (std::isspace(static_cast<unsigned char>(c)) && !in_single && !in_double) {
            if (!cur.empty()) { v.push_back(cur); cur.clear(); }
        } else {
            cur.push_back(c);
        }
    }
    if (!cur.empty()) v.push_back(cur);
    return v;
}

bool copy_file_recursive(const fs::path& src, const fs::path& dst) {
    std::error_code ec;
    if (!fs::exists(src, ec)) return false;
    if (fs::is_directory(src, ec)) {
        fs::create_directories(dst, ec);
        if (ec) return false;
        for (fs::recursive_directory_iterator it(src, ec), end; it != end && !ec; ++it) {
            const fs::path rel = fs::relative(it->path(), src, ec);
            if (ec) return false;
            const fs::path out = dst / rel;
            if (it->is_directory(ec)) {
                fs::create_directories(out, ec);
                if (ec) return false;
            } else if (it->is_symlink(ec)) {
                std::error_code ec2;
                fs::path target = fs::read_symlink(it->path(), ec2);
                if (ec2) return false;
                fs::create_directories(out.parent_path(), ec2);
                if (ec2) return false;
                fs::remove(out, ec2);
                std::filesystem::create_symlink(target, out, ec2);
                if (ec2) return false;
            } else {
                fs::create_directories(out.parent_path(), ec);
                if (ec) return false;
                fs::copy_file(it->path(), out, fs::copy_options::overwrite_existing, ec);
                if (ec) return false;
            }
        }
        return !ec;
    } else {
        fs::create_directories(dst.parent_path(), ec);
        if (ec) return false;
        fs::copy_file(src, dst, fs::copy_options::overwrite_existing, ec);
        return !ec;
    }
}

void ls(const fs::path& p) {
    std::error_code ec;
    if (!fs::exists(p, ec)) { std::cerr << "err: no such file or directory\n"; return; }
    if (fs::is_directory(p, ec)) {
        for (auto& e : fs::directory_iterator(p, ec)) {
            std::cout << e.path().filename().string() << "\n";
        }
        if (ec) std::cerr << "err\n";
    } else {
        std::cout << p.filename().string() << "\n";
    }
}

void search_name(const fs::path& p, const std::string& pat) {
    std::error_code ec;
    if (!fs::exists(p, ec)) { std::cerr << "err\n"; return; }
    for (auto& e : fs::recursive_directory_iterator(p, ec)) {
        if (e.path().filename().string().find(pat) != std::string::npos) {
            std::cout << fs::relative(e.path(), p, ec).string() << "\n";
        }
        if (ec) { std::cerr << "err\n"; return; }
    }
}

mode_t parse_mode(const std::string& s) {
    if (s.empty() || s.size() > 4) return 0;
    char* end = nullptr;
    long m = std::strtol(s.c_str(), &end, 8);
    if (!end || *end != '\0' || m < 0 || m > 07777) return 0;
    return static_cast<mode_t>(m);
}

std::string errno_msg() {
    return std::string(std::strerror(errno));
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::string line;
    while (true) {
        std::cout << cwd() << " > " << std::flush;
        if (!std::getline(std::cin, line)) break;
        auto a = split_quoted(line);
        if (a.empty()) continue;
        const std::string& cmd = a[0];

        if (cmd == "exit") {
            break;
        } else if (cmd == "pwd") {
            std::cout << cwd() << "\n";
        } else if (cmd == "ls") {
            fs::path p = a.size() > 1 ? fs::path(a[1]) : fs::path(cwd());
            ls(p);
        } else if (cmd == "cd") {
            fs::path p;
            if (a.size() > 1) p = fs::path(a[1]);
            else {
                const char* h = std::getenv("HOME");
                p = h ? fs::path(h) : fs::path("/");
            }
            std::error_code ec;
            fs::current_path(p, ec);
            if (ec) std::cerr << "err: " << ec.message() << "\n";
        } else if (cmd == "cp" && a.size() >= 3) {
            fs::path s = a[1], d = a[2];
            if (!copy_file_recursive(s, d)) std::cerr << "err\n";
        } else if (cmd == "mv" && a.size() >= 3) {
            std::error_code ec;
            fs::rename(a[1], a[2], ec);
            if (ec) std::cerr << "err: " << ec.message() << "\n";
        } else if (cmd == "rm" && a.size() >= 2) {
            std::error_code ec;
            fs::path p = a[1];
            if (fs::is_directory(p, ec)) fs::remove_all(p, ec);
            else fs::remove(p, ec);
            if (ec) std::cerr << "err: " << ec.message() << "\n";
        } else if (cmd == "mkdir" && a.size() >= 2) {
            std::error_code ec;
            fs::create_directories(a[1], ec);
            if (ec) std::cerr << "err: " << ec.message() << "\n";
        } else if (cmd == "touch" && a.size() >= 2) {
            std::ofstream f(a[1], std::ios::app);
            if (!f) std::cerr << "err: " << errno_msg() << "\n";
        } else if (cmd == "search" && a.size() >= 2) {
            search_name(fs::path(cwd()), a[1]);
        } else if (cmd == "chmod" && a.size() >= 3) {
            mode_t m = parse_mode(a[1]);
            if (!m) { std::cerr << "err: bad mode\n"; continue; }
            if (::chmod(a[2].c_str(), m) != 0) std::cerr << "err: " << errno_msg() << "\n";
        } else if (cmd == "help") {
            std::cout << "ls [path]\ncd [path]\npwd\ncp <src> <dst>\nmv <src> <dst>\nrm
<path>\nmkdir <path>\ntouch <file>\nsearch <pattern>\nchmod <octal> <path>\nexit\n";
        } else {
            std::cerr << "err\n";
        }
    }
    return 0;
}
```

## Screenshots

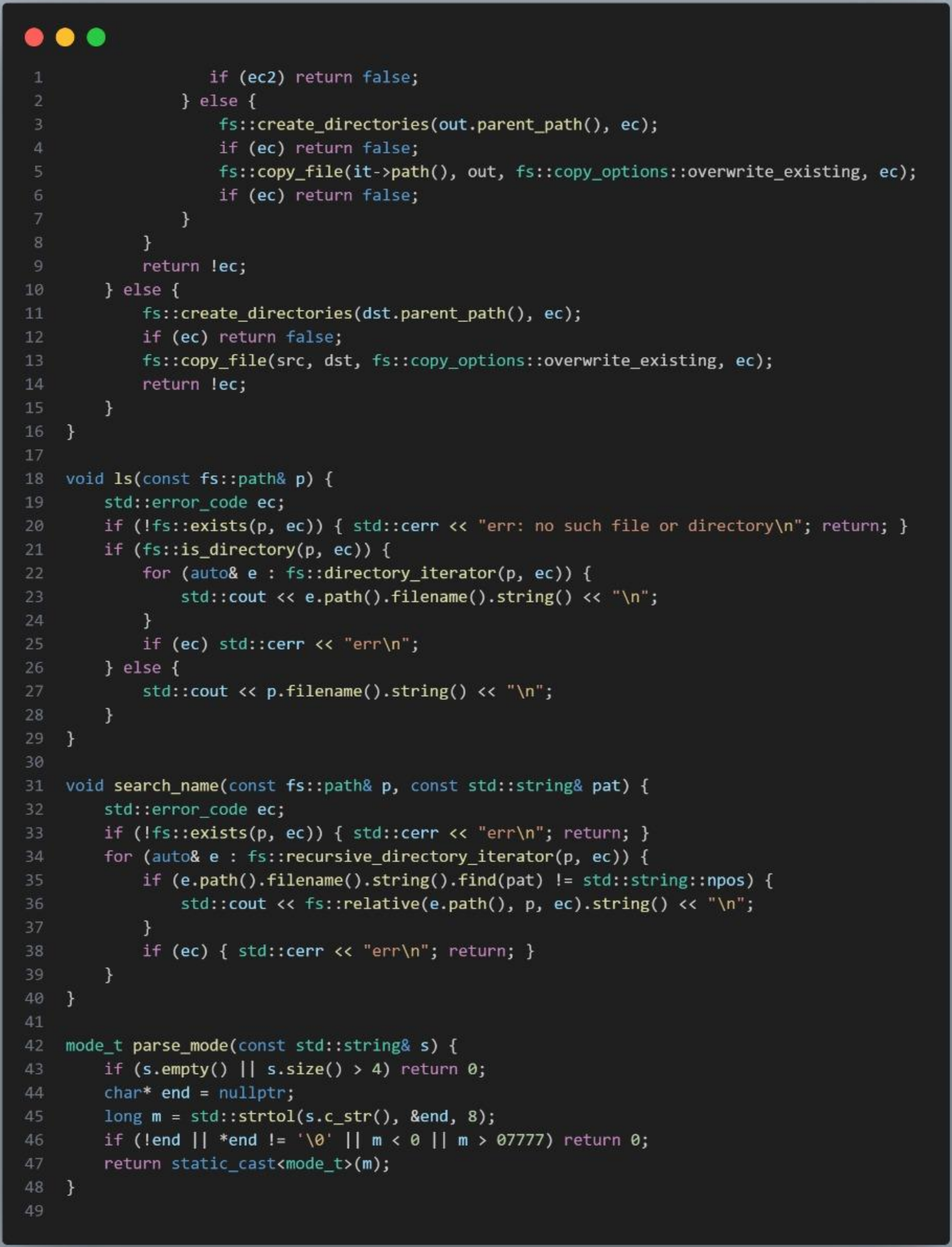
## Code:



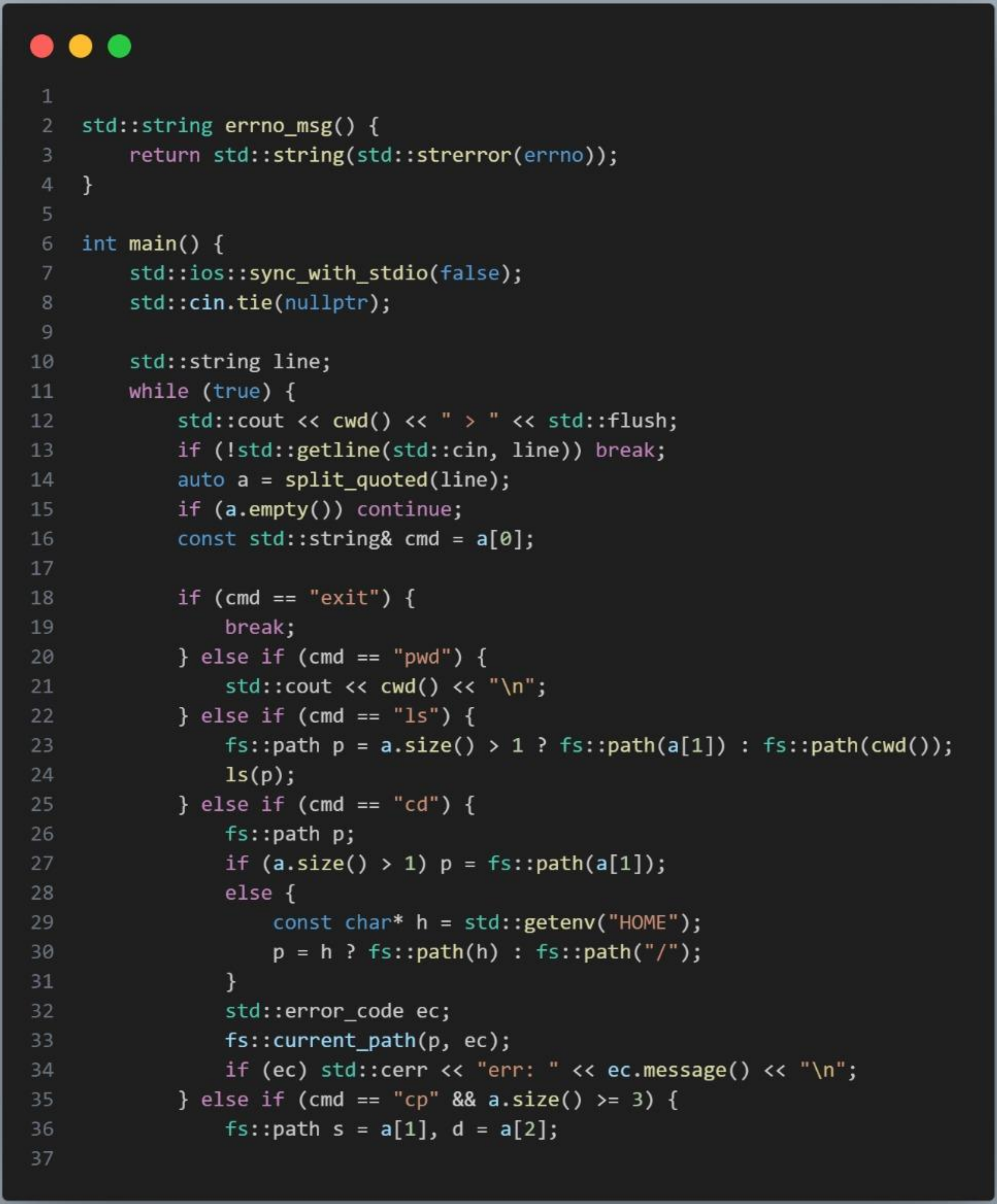
```cpp
#include <filesystem>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <cstdlib>
#include <cerrno>
#include <cstring>
#include <sys/stat.h>
#include <unistd.h>

namespace fs = std::filesystem;

std::string cwd() {
    std::error_code ec;
    auto p = fs::current_path(ec);
    return ec ? std::string{"<unknown>"} : p.string();
}

std::vector<std::string> split_quoted(const std::string& s) {
    std::vector<std::string> v;
    std::string cur;
    bool in_single = false, in_double = false, esc = false;
    for (char c : s) {
        if (esc) { cur.push_back(c); esc = false; continue; }
        if (c == '\\') { esc = true; continue; }
        if (c == '\'' && !in_double) { in_single = !in_single; continue; }
        if (c == '"' && !in_single) { in_double = !in_double; continue; }
        if (std::isspac
```



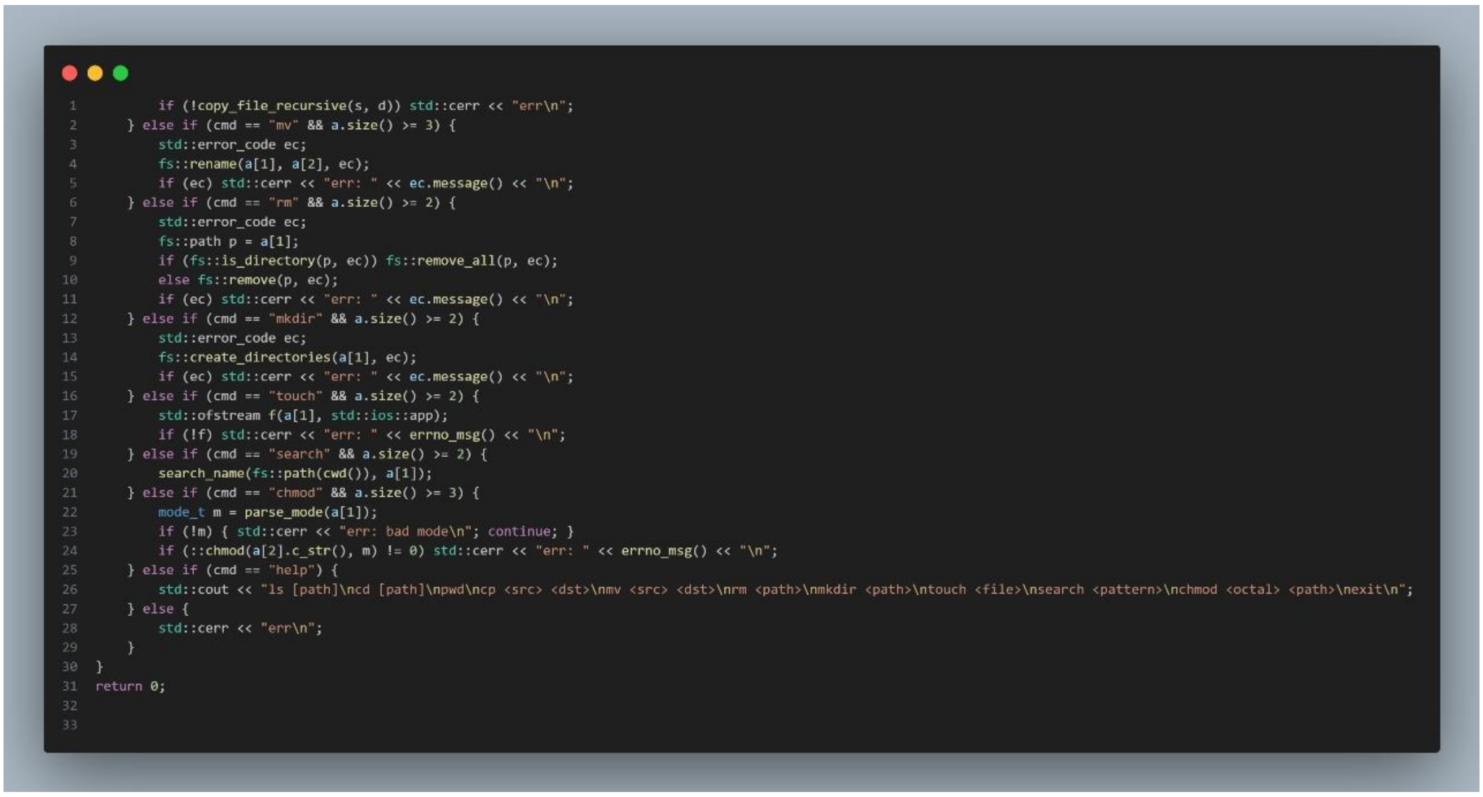
```cpp
        if (std::isspace(static_cast<unsigned char>(c)) && !in_single && !in_double) {
            if (!cur.empty()) { v.push_back(cur); cur.clear(); }
        } else {
            cur.push_back(c);
        }
    }
    if (!cur.empty()) v.push_back(cur);
    return v;
}

bool copy_file_recursive(const fs::path& src, const fs::path& dst) {
    std::error_code ec;
    if (!fs::exists(src, ec)) return false;
    if (fs::is_directory(src, ec)) {
        fs::create_directories(dst, ec);
        if (ec) return false;
        for (fs::recursive_directory_iterator it(src, ec), end; it != end && !ec; ++it) {
            const fs::path rel = fs::relative(it->path(), src, ec);
            if (ec) return false;
            const fs::path out = dst / rel;
            if (it->is_directory(ec)) {
                fs::create_directories(out, ec);
                if (ec) return false;
            } else if (it->is_symlink(ec)) {
                std::error_code ec2;
                fs::path target = fs::read_symlink(it->path(), ec2);
                if (ec2) return false;
                fs::create_directories(out.parent_path(), ec2);
                if (ec2) return false;
                fs::remove(out, ec2);
                std::filesystem::create_symlink(target, out, ec2);
                if (ec2) return false;

```



```cpp
                if (ec2) return false;
            } else {
                fs::create_directories(out.parent_path(), ec);
                if (ec) return false;
                fs::copy_file(it->path(), out, fs::copy_options::overwrite_existing, ec);
                if (ec) return false;
            }
        }
        return !ec;
    } else {
        fs::create_directories(dst.parent_path(), ec);
        if (ec) return false;
        fs::copy_file(src, dst, fs::copy_options::overwrite_existing, ec);
        return !ec;
    }
}

void ls(const fs::path& p) {
    std::error_code ec;
    if (!fs::exists(p, ec)) { std::cerr << "err: no such file or directory\n"; return; }
    if (fs::is_directory(p, ec)) {
        for (auto& e : fs::directory_iterator(p, ec)) {
            std::cout << e.path().filename().string() << "\n";
        }
        if (ec) std::cerr << "err\n";
    } else {
        std::cout << p.filename().string() << "\n";
    }
}

void search_name(const fs::path& p, const std::string& pat) {
    std::error_code ec;
    if (!fs::exists(p, ec)) { std::cerr << "err\n"; return; }
    for (auto& e : fs::recursive_directory_iterator(p, ec)) {
        if (e.path().filename().string().find(pat) != std::string::npos) {
            std::cout << fs::relative(e.path(), p, ec).string() << "\n";
        }
        if (ec) { std::cerr << "err\n"; return; }
    }
}

mode_t parse_mode(const std::string& s) {
    if (s.empty() || s.size() > 4) return 0;
    char* end = nullptr;
    long m = std::strtol(s.c_str(), &end, 8);
    if (!end || *end != '\0' || m < 0 || m > 07777) return 0;
    return static_cast<mode_t>(m);
}

```

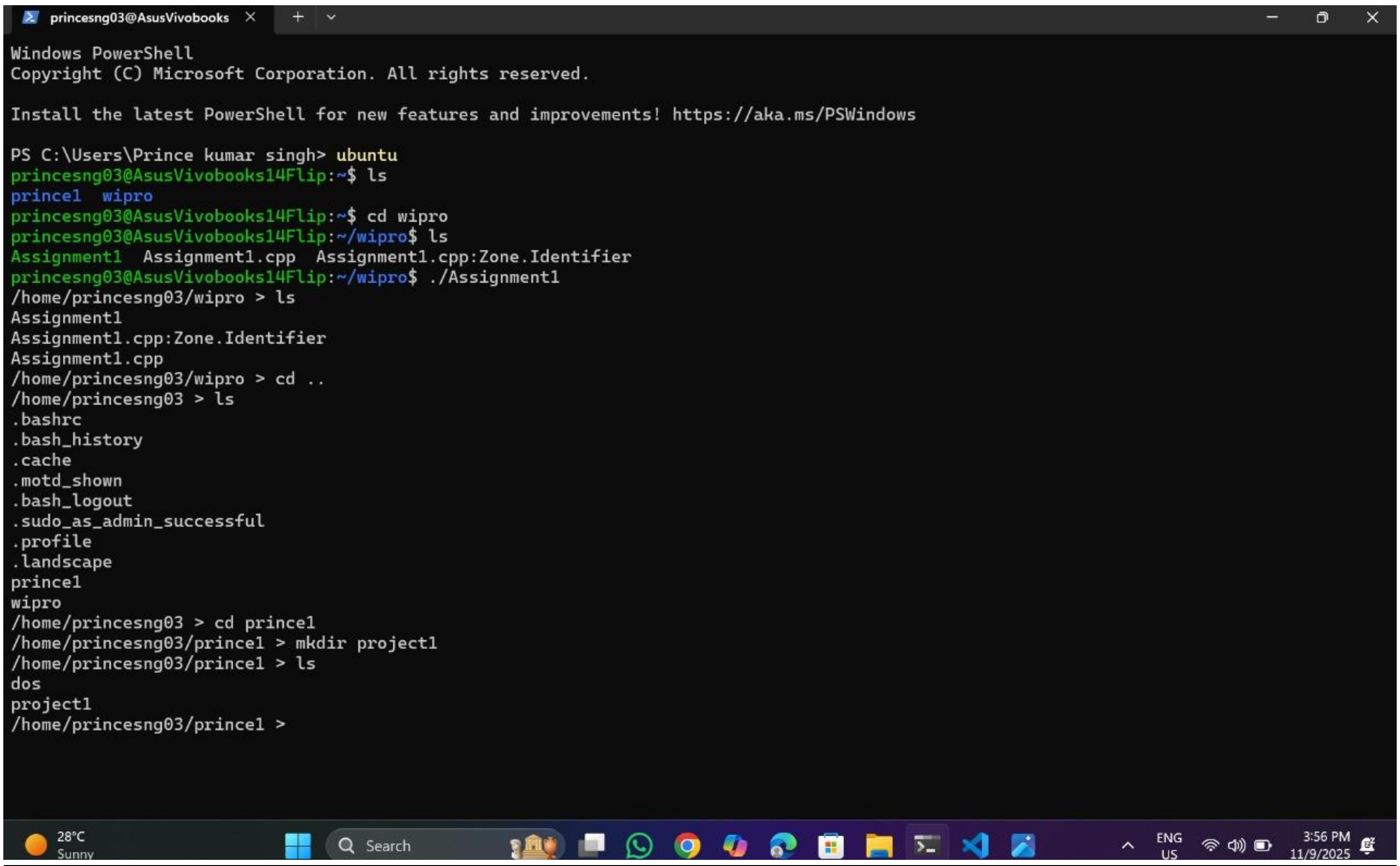


```cpp

std::string errno_msg() {
    return std::string(std::strerror(errno));
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    std::string line;
    while (true) {
        std::cout << cwd() << " > " << std::flush;
        if (!std::getline(std::cin, line)) break;
        auto a = split_quoted(line);
        if (a.empty()) continue;
        const std::string& cmd = a[0];

        if (cmd == "exit") {
            break;
        } else if (cmd == "pwd") {
            std::cout << cwd() << "\n";
        } else if (cmd == "ls") {
            fs::path p = a.size() > 1 ? fs::path(a[1]) : fs::path(cwd());
            ls(p);
        } else if (cmd == "cd") {
            fs::path p;
            if (a.size() > 1) p = fs::path(a[1]);
            else {
                const char* h = std::getenv("HOME");
                p = h ? fs::path(h) : fs::path("/");
            }
            std::error_code ec;
            fs::current_path(p, ec);
            if (ec) std::cerr << "err: " << ec.message() << "\n";
        } else if (cmd == "cp" && a.size() >= 3) {
            fs::path s = a[1], d = a[2];

```



```cpp
        if (!copy_file_recursive(s, d)) std::cerr << "err\n";
    } else if (cmd == "mv" && a.size() >= 3) {
        std::error_code ec;
        fs::rename(a[1], a[2], ec);
        if (ec) std::cerr << "err: " << ec.message() << "\n";
    } else if (cmd == "rm" && a.size() >= 2) {
        std::error_code ec;
        fs::path p = a[1];
        if (fs::is_directory(p, ec)) fs::remove_all(p, ec);
        else fs::remove(p, ec);
        if (ec) std::cerr << "err: " << ec.message() << "\n";
    } else if (cmd == "mkdir" && a.size() >= 2) {
        std::error_code ec;
        fs::create_directories(a[1], ec);
        if (ec) std::cerr << "err: " << ec.message() << "\n";
    } else if (cmd == "touch" && a.size() >= 2) {
        std::ofstream f(a[1], std::ios::app);
        if (!f) std::cerr << "err: " << errno_msg() << "\n";
    } else if (cmd == "search" && a.size() >= 2) {
        search_name(fs::path(cwd()), a[1]);
    } else if (cmd == "chmod" && a.size() >= 3) {
        mode_t m = parse_mode(a[1]);
        if (!m) { std::cerr << "err: bad mode\n"; continue; }
        if (::chmod(a[2].c_str(), m) != 0) std::cerr << "err: " << errno_msg() << "\n";
    } else if (cmd == "help") {
        std::cout << "ls [path]\ncd [path]\npwd\ncp <src> <dst>\nmv <src> <dst>\nrm <path>\nmkdir <path>\ntouch <file>\nsearch <pattern>\nchmod <octal> <path>\nexit\n";
    } else {
        std::cerr << "err\n";
    }
}
return 0;
```

## Output :



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Prince kumar singh> ubuntu
princesng03@AsusVivobooks14Flip:~$ ls
prince1  wipro
princesng03@AsusVivobooks14Flip:~$ cd wipro
princesng03@AsusVivobooks14Flip:~/wipro$ ls
Assignment1  Assignment1.cpp  Assignment1.cpp:Zone.Identifier
princesng03@AsusVivobooks14Flip:~/wipro$ ./Assignment1
/home/princesng03/wipro > ls
Assignment1
Assignment1.cpp:Zone.Identifier
Assignment1.cpp
/home/princesng03/wipro > cd ..
/home/princesng03 > ls
.bashrc
.bash_history
.cache
.motd_shown
.bash_logout
.sudo_as_admin_successful
.profile
.landscape
prince1
wipro
/home/princesng03 > cd prince1
/home/princesng03/prince1 > mkdir project1
/home/princesng03/prince1 > ls
dos
project1
/home/princesng03/prince1 >
```