Assignment 1 : File Explorer

```
in11@LAPTOP-FK814BKI:/mnt/c/Users/Win11/Desktop/WiproProject$ cd Desktop
-bash: cd: Desktop: No such file or directory
in11@LAPTOP-FK814BKI:/mnt/c/Users/Win11/Desktop/WiproProject$ g++ -std=c++17 file_Explorer.cpp -o explorer
in11@LAPTOP-FK814BKI:/mnt/c/Users/Win11/Desktop/WiproProject$ ./explorer
Simple File Explorer (C++ / Linux). Type 'help' for commands.

Commands:
  ls [path]                - List directory (current dir if omitted)
  pwd                      - Show current directory
  cd <dir>                 - Change directory (relative or absolute)
  back                     - Go to parent directory
  copy <src> <dest>        - Copy file/directory (recursive)
  move <src> <dest>        - Move (rename) file/directory
  rm <path>                - Delete file or directory (recursive)
  mkdir <dir>              - Create directory
  touch <file>             - Create empty file (like touch)
  search <name> [start_dir] - Recursively search for filename (or partial match)
  perms <path>             - Show owner/group/others rwx permissions
  chmod <path> <xyz>       - Set permissions using octal (e.g. 755)
  help                     - Show this help
  exit                     - Exit program
```

```
["/mnt/c/Users/Win11/Desktop/WiproProject"]> ls
[FILE] explorer  rwx rwx rwx
[FILE] explorer.exe  rwx rwx rwx
[FILE] file_Explorer.cpp  rwx rwx rwx
[FILE] test.txt  rwx rwx rwx
[FILE] test2.txt  rwx rwx rwx
```

```
["/mnt/c/Users/Win11/Desktop/WiproProject"]> touch test1.txt

["/mnt/c/Users/Win11/Desktop/WiproProject"]> ls
[FILE] explorer  rwx rwx rwx
[FILE] explorer.exe  rwx rwx rwx
[FILE] file_Explorer.cpp  rwx rwx rwx
[FILE] test.txt  rwx rwx rwx
[FILE] test1.txt  rwx rwx rwx
[FILE] test2.txt  rwx rwx rwx

["/mnt/c/Users/Win11/Desktop/WiproProject"]>
```

Code:

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

namespace fs = std::filesystem;

void show_help() {
   std::cout << R"(
Commands:
 ls [path]              - List directory (current dir if omitted)
 pwd                    - Show current directory
 cd <dir>               - Change directory (relative or absolute)
 back                   - Go to parent directory
 copy <src> <dest>      - Copy file/directory (recursive)
 move <src> <dest>      - Move (rename) file/directory
 rm <path>              - Delete file or directory (recursive)
 mkdir <dir>            - Create directory
 touch <file>           - Create empty file (like touch)
 search <name> [start_dir] - Recursively search for filename (or partial match)
 perms <path>           - Show owner/group/others rwx permissions
 chmod <path> <xyz>     - Set permissions using octal (e.g. 755)
 help                   - Show this help
 exit                   - Exit program
)";
}
```

```
std::string perms_to_string(fs::perms p) {
    auto check = [&](fs::perms bit)->char {
        return ( (p & bit) != fs::perms::none ) ? 'r' : '-';
    };

    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 += ' ';
    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 += ' ';
    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;
}

void do_ls(const std::string &arg) {
    fs::path target = arg.empty() ? fs::current_path() : fs::path(arg);
    try {
        if (!fs::exists(target)) {
            std::cout << "Path does not exist: " << target << "\n";
            return;
        }
        if (!fs::is_directory(target)) {
            std::cout << target << " is not a directory\n";
            return;
        }
        for (auto &entry : fs::directory_iterator(target)) {
```

```
            std::string type;
            try {
                if (fs::is_directory(entry.symlink_status())) type = "[DIR] ";
                else if (fs::is_regular_file(entry.symlink_status())) type = "[FILE]";
                else type = "[OTHER]";
            } catch(...) { type = "[?]"; }
            std::string name = entry.path().filename().string();
            //permissions
            std::string pstr = "????";
            try {
                pstr = perms_to_string(fs::status(entry.path()).permissions());
            } catch(...) { pstr = "---- ---- ---"; }
            std::cout << type << " " << name << "  " << pstr << "\n";
        }
    } catch (const std::exception &e) {
        std::cout << "ls error: " << e.what() << "\n";
    }
}

void do_cd(const std::string &dir) {
    fs::path p = dir.empty() ? fs::current_path() : fs::path(dir);
    try {
        if (!fs::exists(p) || !fs::is_directory(p)) {
            std::cout << "Not a directory: " << p << "\n";
            return;
        }
        fs::current_path(p);
    } catch (const std::exception &e) {
        std::cout << "cd error: " << e.what() << "\n";
    }
}

void do_copy(const std::string &src, const std::string &dest) {
```

```
    try {
        fs::path s = src, d = dest;
        if (!fs::exists(s)) { std::cout << "Source does not exist\n"; return; }
        if (fs::is_directory(s)) {
            fs::copy(s, d, fs::copy_options::recursive | fs::copy_options::overwrite_existing);
        } else {
            fs::create_directories(d.parent_path());
            fs::copy_file(s, d, fs::copy_options::overwrite_existing);
        }
    } catch (const std::exception &e) {
        std::cout << "copy error: " << e.what() << "\n";
    }
}

void do_move(const std::string &src, const std::string &dest) {
    try {
        fs::rename(src, dest);
    } catch (const std::exception &e) {
        std::cout << "move error: " << e.what() << "\n";
    }
}

void do_rm(const std::string &path) {
    try {
        if (!fs::exists(path)) { std::cout << "Path does not exist\n"; return; }
        fs::remove_all(path);
    } catch (const std::exception &e) {
        std::cout << "rm error: " << e.what() << "\n";
    }
}

void do_mkdir(const std::string &dir) {
    try {
```

```
        if (fs::exists(dir)) { std::cout << "Already exists\n"; return; }
        fs::create_directories(dir);
    } catch (const std::exception &e) {
        std::cout << "mkdir error: " << e.what() << "\n";
    }
}

void do_touch(const std::string &file) {
    try {
        std::ofstream ofs(file, std::ios::app);
        // update timestamp by closing and reopening
        ofs.close();
    } catch (const std::exception &e) {
        std::cout << "touch error: " << e.what() << "\n";
    }
}

void search_recursive(const fs::path &start, const std::string &name) {
    try {
        for (auto it = fs::recursive_directory_iterator(start, fs::directory_options::skip_permission_denied);
             it != fs::recursive_directory_iterator(); ++it) {
            try {
                std::string fname = it->path().filename().string();
                if (fname.find(name) != std::string::npos) {
                    std::cout << "Found: " << it->path() << "\n";
                }
            } catch(...) { /* skip entries we cannot access */ }
        }
    } catch (const std::exception &e) {
        std::cout << "search error: " << e.what() << "\n";
    }
}
```

```
void show_perms(const std::string &path) {
    try {
        if (!fs::exists(path)) { std::cout << "No such file\n"; return; }
        fs::perms p = fs::status(path).permissions();
        std::cout << "Permissions: " << perms_to_string(p) << "\n";
    } catch (const std::exception &e) {
        std::cout << "perms error: " << e.what() << "\n";
    }
}

void do_chmod(const std::string &path, const std::string &octal) {
    try {
        if (!fs::exists(path)) { std::cout << "No such file\n"; return; }
        if (octal.size() != 3 || !std::all_of(octal.begin(), octal.end(), ::isdigit)) {
            std::cout << "chmod expects a 3-digit octal like 755\n";
            return;
        }
        auto digit = [&](int i) { return octal[i] - '0'; };
        fs::perms p = fs::perms::none;

        int od = digit(0), gd = digit(1), td = digit(2);
        auto set_from_digit = [&](int d, fs::perms r, fs::perms w, fs::perms x) {
            if (d & 4) p |= r;
            if (d & 2) p |= w;
            if (d & 1) p |= x;
        };
        set_from_digit(od, fs::perms::owner_read, fs::perms::owner_write, fs::perms::owner_exec);
        set_from_digit(gd, fs::perms::group_read, fs::perms::group_write, fs::perms::group_exec);
        set_from_digit(td, fs::perms::others_read, fs::perms::others_write, fs::perms::others_exec);

        fs::permissions(path, p, fs::perm_options::replace);
    } catch (const std::exception &e) {
        std::cout << "chmod error: " << e.what() << "\n";
```

```
    }
}

std::vector<std::string> split_args(const std::string &line) {
    std::istringstream iss(line);
    std::vector<std::string> out;
    std::string tok;
    while (iss >> std::quoted(tok) || iss >> tok) {
        out.push_back(tok);
    }
    return out;
}

int main() {
    std::cout << "Simple File Explorer (C++ / Linux). Type 'help' for commands.\n";
    show_help();

    std::string line;
    while (true) {
        try {
            std::cout << "\n[" << fs::current_path() << "]> ";
        } catch(...) {
            std::cout << "\n[unknown]> ";
        }
        if (!std::getline(std::cin, line)) break;
        if (line.empty()) continue;


        std::vector<std::string> args = split_args(line);
        if (args.empty()) continue;
        std::string cmd = args[0];

        if (cmd == "exit") break;
```

```
else if (cmd == "help") show_help();
else if (cmd == "ls") {
    if (args.size() >= 2) do_ls(args[1]);
    else do_ls("");
}
else if (cmd == "pwd") {
    try { std::cout << fs::current_path() << "\n"; } catch(...) { std::cout << "?\n"; }
}
else if (cmd == "cd") {
    if (args.size() >= 2) do_cd(args[1]);
    else std::cout << "cd requires a directory argument\n";
}
else if (cmd == "back") {
    try { fs::current_path(fs::current_path().parent_path()); } catch(...) { std::cout << "Cannot go back\n"; }
}
else if (cmd == "copy") {
    if (args.size() >= 3) do_copy(args[1], args[2]);
    else std::cout << "Usage: copy <src> <dest>\n";
}
else if (cmd == "move") {
    if (args.size() >= 3) do_move(args[1], args[2]);
    else std::cout << "Usage: move <src> <dest>\n";
}
else if (cmd == "rm") {
    if (args.size() >= 2) do_rm(args[1]);
    else std::cout << "Usage: rm <path>\n";
}
else if (cmd == "mkdir") {
    if (args.size() >= 2) do_mkdir(args[1]);
    else std::cout << "Usage: mkdir <dir>\n";
}
else if (cmd == "touch") {
    if (args.size() >= 2) do_touch(args[1]);
```

```
            else std::cout << "Usage: touch <file>\n";
        }
        else if (cmd == "search") {
            if (args.size() >= 2) {
                fs::path start = (args.size() >= 3) ? fs::path(args[2]) : fs::current_path();
                search_recursive(start, args[1]);
            } else std::cout << "Usage: search <name> [start_dir]\n";
        }
        else if (cmd == "perms") {
            if (args.size() >= 2) show_perms(args[1]);
            else std::cout << "Usage: perms <path>\n";
        }
        else if (cmd == "chmod") {
            if (args.size() >= 3) do_chmod(args[1], args[2]);
            else std::cout << "Usage: chmod <path> <3-digit-octal>\n";
        }
        else {
            std::cout << "Unknown command. Type 'help' for list of commands.\n";
        }
    }

    std::cout << "Goodbye.\n";
    return 0;
}
```