ZYF AXL ALRAFI DIGA

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#ifdef \_\_unix\_\_

#include <dirent.h>

#include <sys/stat.h>

#include <unistd.h>

#include <limits.h>

#elif \_WIN32

#include <windows.h>

#include <string>

#endif

void listFiles(const std::string& directoryPath, int listOption) {

std::vector<std::string> files;

#ifdef \_\_unix\_\_

DIR\* dir = opendir(directoryPath.c\_str());

if (dir == nullptr) {

std::cerr << "Error opening directory: " << directoryPath << std::endl;

return;

}

struct dirent\* entry;

while ((entry = readdir(dir)) != nullptr) {

if (entry->d\_name[0] != '.') {

files.push\_back(entry->d\_name);

}

}

closedir(dir);

#elif \_WIN32

std::string searchPath = directoryPath + "\\\*";

WIN32\_FIND\_DATA findFileData;

HANDLE hFind = FindFirstFile(searchPath.c\_str(), &findFileData);

if (hFind == INVALID\_HANDLE\_VALUE) {

std::cerr << "Error opening directory: " << directoryPath << std::endl;

return;

}

do {

if (findFileData.cFileName[0] != '.') {

files.push\_back(findFileData.cFileName);

}

} while (FindNextFile(hFind, &findFileData) != 0);

FindClose(hFind);

#endif

if (listOption == 1) {

for (const auto& file : files) {

std::cout << file << std::endl;

}

} else if (listOption == 2) {

std::string extension;

std::cout << "Enter file extension (e.g., txt): ";

std::cin >> extension;

for (const auto& file : files) {

if (file.substr(file.find\_last\_of('.') + 1) == extension) {

std::cout << file << std::endl;

}

}

} else if (listOption == 3) {

std::string pattern;

std::cout << "Enter file name pattern (e.g., moha\*.\*): ";

std::cin >> pattern;

for (const auto& file : files) {

if (file.find(pattern) != std::string::npos) {

std::cout << file << std::endl;

}

}

} else {

std::cerr << "Invalid option!" << std::endl;

}

}

void createDirectory(const std::string& directoryPath) {

#ifdef \_\_unix\_\_

if (mkdir(directoryPath.c\_str(), 0755) != 0) {

std::cerr << "Error creating directory: " << directoryPath << std::endl;

} else {

std::cout << "Directory created: " << directoryPath << std::endl;

}

#elif \_WIN32

if (CreateDirectory(directoryPath.c\_str(), nullptr) || GetLastError() == ERROR\_ALREADY\_EXISTS) {

std::cout << "Directory created: " << directoryPath << std::endl;

} else {

std::cerr << "Error creating directory: " << directoryPath << std::endl;

}

#endif

}

void changeDirectory(const std::string& directoryPath) {

#ifdef \_\_unix\_\_

if (chdir(directoryPath.c\_str()) != 0) {

std::cerr << "Error changing directory to: " << directoryPath << std::endl;

} else {

std::cout << "Changed working directory to: " << directoryPath << std::endl;

}

#elif \_WIN32

if (SetCurrentDirectory(directoryPath.c\_str())) {

std::cout << "Changed working directory to: " << directoryPath << std::endl;

} else {

std::cerr << "Error changing directory to: " << directoryPath << std::endl;

}

#endif

}

void listFilesMenu(const std::string& directoryPath) {

int listOption;

std::cout << "LIST FILE DETAIL\n";

std::cout << "1. List All Files\n";

std::cout << "2. List of Extension Files\n";

std::cout << "3. List of Name Pattern Files\n";

std::cout << "Enter the Number: ";

std::cin >> listOption;

std::cin.ignore();

listFiles(directoryPath, listOption);

}

void changeDirectoryMenu() {

int changeOption;

std::string directoryPath;

#ifdef \_\_unix\_\_

char cwd[PATH\_MAX];

if (getcwd(cwd, sizeof(cwd)) != nullptr) {

std::cout << "Current Directory: " << cwd << std::endl;

}

#elif \_WIN32

char cwd[MAX\_PATH];

if (GetCurrentDirectory(MAX\_PATH, cwd)) {

std::cout << "Current Directory: " << cwd << std::endl;

}

#endif

std::cout << "Change Directory Menu\n";

std::cout << "1. Step by Step Backward\n";

std::cout << "2. Goto Root Directory\n";

std::cout << "3. Forward Directory\n";

std::cout << "Enter the Number: ";

std::cin >> changeOption;

std::cin.ignore();

switch (changeOption) {

case 1: {

#ifdef \_\_unix\_\_

char cwd[PATH\_MAX];

if (getcwd(cwd, sizeof(cwd)) != nullptr) {

std::string currentPath(cwd);

size\_t pos = currentPath.find\_last\_of('/');

if (pos != std::string::npos) {

currentPath = currentPath.substr(0, pos);

changeDirectory(currentPath);

}

}

#elif \_WIN32

char cwd[MAX\_PATH];

if (GetCurrentDirectory(MAX\_PATH, cwd)) {

std::string currentPath(cwd);

size\_t pos = currentPath.find\_last\_of("\\");

if (pos != std::string::npos) {

currentPath = currentPath.substr(0, pos);

changeDirectory(currentPath);

}

}

#endif

break;

}

case 2: {

#ifdef \_\_unix\_\_

changeDirectory("/");

#elif \_WIN32

changeDirectory("C:\\");

#endif

break;

}

case 3: {

std::cout << "Please enter the Directory Name: ";

std::getline(std::cin, directoryPath);

#ifdef \_\_unix\_\_

std::string currentPath;

char cwd[PATH\_MAX];

if (getcwd(cwd, sizeof(cwd)) != nullptr) {

currentPath = cwd;

}

if (directoryPath[0] != '/') {

directoryPath = currentPath + "/" + directoryPath;

}

#elif \_WIN32

std::string currentPath;

char cwd[MAX\_PATH];

if (GetCurrentDirectory(MAX\_PATH, cwd)) {

currentPath = cwd;

}

if (directoryPath[1] != ':') {

directoryPath = currentPath + "\\" + directoryPath;

}

#endif

changeDirectory(directoryPath);

break;

}

default:

std::cerr << "Invalid option!" << std::endl;

break;

}

}

int main() {

int mainOption;

std::string directoryPath;

do {

std::cout << "MAIN MENU\n";

std::cout << "1. To Display List of Files\n";

std::cout << "2. To Create New Directory\n";

std::cout << "3. To Change the Working Directory\n";

std::cout << "4. Exit\n";

std::cout << "Enter the Number: ";

std::cin >> mainOption;

std::cin.ignore();

switch (mainOption) {

case 1: {

std::cout << "Enter the directory path to list files: ";

std::getline(std::cin, directoryPath);

listFilesMenu(directoryPath);

break;

}

case 2: {

std::cout << "Enter the directory path to create: ";

std::getline(std::cin, directoryPath);

createDirectory(directoryPath);

break;

}

case 3: {

changeDirectoryMenu();

break;

}

case 4:

std::cout << "Exiting...\n";

break;

default:

std::cerr << "Invalid option! Please try again.\n";

break;

}

std::cout << "Press any key to continue...";

std::cin.get();

std::cin.ignore();

} while (mainOption != 4);

return 0;

}