

Міністерство освіти і науки України
Національний університет
«Львівська політехніка»
Кафедра електронних обчислювальних
машин

КУРСОВИЙ ПРОЄКТ

з дисципліни «Системне програмне забезпечення»
на тему: Розробка програмного забезпечення для керування
фотофайлами.

Розробка програми, що дозволяє керувати фотофайлами та
виконувати різні дії з їх обробкою на комп'ютері з операційною
системою Windows.

Студента 3-го курсу групи КІ-307

«Комп'ютерна інженерія»

Чіпко М.Б.

Керівник доцент каф. ЕОМ, канд. техн. наук

Олексів М.В.

Національна шкала: _____

Кількість балів: _____

Оцінка ECTS: _____

Члени комісії: _____ _Олексів М.В._
(підпис) (прізвище та ініціали)

Львів 2024

ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ

Основне завдання на курсовий проект полягає у розробці програмного забезпечення "Файловий менеджер", призначеного для ефективного управління файлами та каталогами на операційній системі Windows. Програма забезпечує реалізацію різноманітних операцій, таких як створення, перегляд, видалення, перейменування файлів і каталогів, а також запис даних у файли через інтерфейс командного рядка.

Метою цього проекту є набуття практичних навичок у розробці системних програм з використанням мови програмування C++ та Windows API. Завдання проекту включає аналіз існуючих підходів до управління файлами, розробку архітектури програми та реалізацію функціональних можливостей, що включають створення, видалення, перегляд і перейменування файлів та каталогів.

Завдання проекту:

- Аналіз існуючих варіантів рішення питання
- Розробка архітектури програми
- Реалізація функціональних можливостей
 - a. Створення директорії (createDirectory): виконує створення нової директорії за заданим шляхом.
 - b. Створення файлу (createFile): створює новий файл за заданим шляхом.
 - c. Видалення директорії (deleteDirectory): видаляє директорію за заданим шляхом.
 - d. Видалення файлу (deleteFile): видаляє файл за заданим шляхом.
 - e. Відкриття вмісту директорії (openDirectory): відкриває вміст директорії за заданим шляхом (можливо, в іншому вікні або наступному кроці).
 - f. Відкриття файлу в новому вікні (openFileInNewWindow): відкриває файл у новому вікні або додатку.
 - g. Перейменування файлу або директорії (renameFileOrFolder): перейменовує вказаний файл або директорію.
 - h. Вихід з програми (quit): Вибір опції "Exit" приводить до

встановлення флагу quit, що завершує цикл інтерфейсу.

Очікувані результати:

Результатом розробки програмного забезпечення має стати програма, здатність значно спростити управління файлами та каталогами у середовищі Windows. Інтуїтивно зрозумілий інтерфейс та широкий набір функцій роблять програму корисною як для системних адміністраторів, так і для звичайних користувачів, які віддають перевагу командному рядку. Програма дозволяє ефективно виконувати повсякденні задачі з управління файлами, забезпечуючи стабільну та продуктивну роботу операційної системи.

АНОТАЦІЯ:

Цей проект присвячений розробці системної утиліти "Файловий менеджер", що реалізована на мові програмування C++. Основною метою проекту є створення інструменту, який надає користувачам можливість ефективно управляти файлами та каталогами через інтерфейс командного рядка. Програма забезпечує виконання різноманітних операцій, включаючи створення, перегляд, видалення та перейменування файлів і каталогів, а також запис даних у файли.

Програма використовує клас `FileManager` для структуризації коду та забезпечення гнучкості і розширюваності. Зокрема, функції `createFile` та `createDirectory` відповідають за створення файлів та каталогів, а функції `deleteFile` та `deleteDirectory` — за їх видалення. Для відображення вмісту каталогів використовується функція `listDirectoryContents`, а функція `readFile` дозволяє виводити вміст файлів на екран. Додатково, функція `renameFileOrDirectory` реалізує перейменування файлів та каталогів, що спрощує управління файловою системою.

Для роботи з файловою системою та взаємодії з користувачем використовується бібліотека Windows API, що забезпечує надійне та швидке виконання операцій. Програма також підтримує функцію відкриття файлів у новому вікні, що реалізується за допомогою системних команд.

Результати роботи програми демонструють її здатність значно спростити управління файлами та каталогами у середовищі Windows. Інтуїтивно зрозумілий інтерфейс та широкий набір функцій роблять програму корисною як для системних адміністраторів, так і для звичайних користувачів, які віддають перевагу командному рядку. Програма дозволяє ефективно виконувати повсякденні задачі з управління файлами, забезпечуючи стабільну та продуктивну роботу операційної системи.

Зміст

| | |
|--|-----------|
| ЗАВДАННЯ НА КУРСОВИЙ ПРОЕКТ | 2 |
| Завдання проекту: | 2 |
| Очікувані результати: | 3 |
| АНОТАЦІЯ: | 4 |
| ВСТУП: | 7 |
| РОЗДІЛ 1.АНАЛІТИЧНИЙ огляд | 9 |
| 1. Аналіз методів для керування файловою системою | 9 |
| 2. Огляд існуючих розробок | 10 |
| 3. Цільова аудиторія | 11 |
| 4. Висновок до розділу “Аналітичний огляд” | 12 |
| РОЗДІЛ 2. ПРОЕКТУВАННЯ | 12 |
| Функціональні вимоги | 13 |
| Нефункціональні вимоги | 15 |
| 2.2 Розробка структурної схеми | 17 |
| 2.3 Вибір засобів реалізації | 19 |
| 2.4 Висновок до розділу “Проектування” | 21 |
| РОЗДІЛ 3. Реалізація | 22 |
| 3.1 Високорівневий опис реалізації | 22 |
| Основні функціональність: | 22 |
| Технічні аспекти реалізації: | 23 |
| 3.2 Реалізації дизайну | 23 |
| 1. Консольне меню | 23 |
| 2. Кольорове виділення тексту | 24 |
| 3. Інтерактивне введення та виведення | 24 |
| 4. Використання системних команд | 24 |
| 5. Обробка помилок та винятків | 24 |
| 3.3 Опис процесу застосунку | 25 |
| 1. Ініціалізація | 25 |

| | |
|---|----|
| 2. Основний цикл програми (main функція) | 25 |
| 3. Взаємодія з користувачем | 26 |
| 3.4 Висновок до розділу “Реалізація” | 26 |
| РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... | 27 |
| 4.1 Тестування роботи | 27 |
| 1. Функціональне тестування | 27 |
| 2. Тестування продуктивності | 32 |
| 3. Обробка помилок | 32 |
| Загальний підхід..... | 33 |
| 4.2 Висновок до розділу “Тестування програмного забезпечення” | 33 |
| Список використаних джерел | 36 |
| Додаток | 38 |

ВСТУП:

Розробка системної утиліти "Файловий менеджер" є важливою задачею в контексті сучасних інформаційних технологій. У наш час комп'ютери є невід'ємною частиною повсякденного життя, і ефективне управління файлами та каталогами стає критично важливим для користувачів усіх рівнів, від початківців до досвідчених системних адміністраторів. Цей курсовий проект присвячений створенню утиліти, яка дозволяє користувачам здійснювати основні операції з файлами та каталогами через інтерфейс командного рядка.

Однією з ключових цілей проекту є розробка інструменту, що забезпечує інтуїтивно зрозумілий інтерфейс для виконання повсякденних завдань з управління файлами, таких як створення, перегляд, видалення, перейменування та запис даних у файли. Утиліта, написана на мові програмування C++, використовує бібліотеку Windows API для забезпечення швидкої та надійної роботи. Вибір C++ обумовлений його потужністю та гнучкістю, що дозволяє реалізувати ефективні алгоритми та забезпечити високу продуктивність додатку.

Перш за все, варто зазначити, що робота з файлами та каталогами є однією з найпоширеніших задач у користувачів операційної системи Windows. Будь-який користувач, незалежно від рівня підготовки, стикається з необхідністю створення нових файлів та папок, видалення застарілих або непотрібних даних, перейменування файлів для кращої організації та структурування даних. Таким чином, утиліта "Файловий менеджер" повинна бути максимально зручною та зрозумілою у використанні, забезпечуючи широкий спектр функцій для управління файлами.

Основні функціональні можливості утиліти включають створення

файлів та каталогів, їх перегляд, видалення та перейменування. Для реалізації цих функцій програма використовує різноманітні системні виклики Windows API, що дозволяє ефективно взаємодіяти з файловою системою. Наприклад, функція `CreateFileA` використовується для створення нових файлів, `DeleteFileA` — для їх видалення, а `CreateDirectory` — для створення нових каталогів. Для відображення вмісту каталогів використовується функція `FindFirstFile` та `FindNextFile`, що дозволяє користувачам швидко переглядати вміст директорій.

Однією з особливостей утиліти є підтримка роботи з командним рядком, що дозволяє користувачам виконувати всі операції, не покидаючи консолі. Це особливо важливо для системних адміністраторів та просунутих користувачів, які віддають перевагу швидкому та ефективному способу управління файлами без використання графічного інтерфейсу. Крім того, командний рядок дозволяє автоматизувати багато завдань за допомогою сценаріїв, що робить утиліту ще більш потужною та гнучкою.

Для забезпечення зручності використання утиліта має просту та зрозумілу структуру меню, що дозволяє швидко знаходити необхідні функції. Вибір варіантів здійснюється за допомогою клавіш зі стрілками, а підтвердження дії — натисканням клавіші Enter. Такий підхід забезпечує інтуїтивно зрозумілу навігацію та мінімізує час на освоєння утиліти.

Зокрема, для створення файлів та каталогів користувачам необхідно лише ввести шлях та ім'я нового файлу або каталогу, після чого утиліта автоматично створює їх у зазначеному місці. Для видалення файлів або каталогів користувачам достатньо вказати шлях до об'єкта, який вони хочуть видалити, і утиліта виконає необхідні дії. Перейменування файлів та каталогів також здійснюється дуже просто — користувачам необхідно вказати поточне ім'я об'єкта та нове ім'я, після чого утиліта оновить назву.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

У цьому аналітичному огляді розглянуто методи та інструменти для керування файловою системою в операційній системі Windows. Огляд включає аналіз стандартних інструментів, таких як Провідник Windows, командний рядок і PowerShell, а також системних викликів (API) і сторонніх програм, таких як Total Commander, FreeCommander і Explorer++. Описуються можливості кожного інструменту, їх переваги та недоліки, а також їх цільова аудиторія, яка включає звичайних користувачів, просунутих користувачів та системних адміністраторів і розробників. Огляд завершується висновком про широкий спектр доступних методів керування файловою системою, який дозволяє вибрати оптимальний інструмент для конкретних потреб користувача.

1. Аналіз методів для керування файловою системою

Керування файловою системою є однією з основних задач, які виникають при розробці програмного забезпечення для операційних систем. Файлова система забезпечує організацію, зберігання, доступ і керування даними на носіях інформації. У сучасних операційних системах, таких як Windows, існує кілька методів керування файловою системою:

Стандартні інструменти ОС:

- **Провідник Windows (Windows Explorer):** Забезпечує графічний інтерфейс для перегляду та керування файлами та папками.
- **Командний рядок (Command Prompt):** Дозволяє виконувати різноманітні операції з файлами та папками за допомогою команд, таких як `dir`, `copy`, `del`, `move`, `mkdir`, тощо.

- **PowerShell:** Потужний інструмент для автоматизації завдань адміністрування системи та керування файлами за допомогою скриптів.

Системні виклики (API):

- **Windows API:** Забезпечує набір функцій для розробки програмного забезпечення, яке взаємодіє з файловою системою. Основні функції включають `CreateFile`, `ReadFile`, `WriteFile`, `DeleteFile`, `MoveFile`, тощо.
- **.NET Framework:** Містить класи, такі як `System.IO.File`, `System.IO.Directory`, які спрощують роботу з файлами та директоріями.

Сторонні інструменти та бібліотеки:

- **Total Commander, FreeCommander:** Програми для керування файлами з розширеним функціоналом порівняно зі стандартним Провідником Windows.
- **LibGit2, TortoiseSVN:** Бібліотеки та інструменти для керування версіями файлів, інтеграції з системами контролю версій.

2. Огляд існуючих розробок

Існує багато програм та утиліт, що надають розширені можливості для керування файловою системою:

- **Total Commander:**
 - Дозволяє виконувати різноманітні операції з файлами, такі як копіювання, переміщення, видалення, архівування.
 - Підтримує плагіни для розширення функціональності.
 - Має двохпанельний інтерфейс, що спрощує роботу з файлами.
- **FreeCommander:**
 - Безкоштовний файловий менеджер з великою кількістю функцій.

- Підтримує перегляд та редагування файлів, порівняння каталогів, синхронізацію папок.
- Має зручний інтерфейс і підтримку гарячих клавіш.
- **Explorer++:**
 - Легка і швидка альтернатива стандартному Провіднику Windows.
 - Підтримує вкладки для зручної роботи з декількома каталогами одночасно.
 - Має відкритий вихідний код, що дозволяє розробникам вносити власні зміни та покращення.
- **XYplorer:**
 - Комерційний файловий менеджер з великим набором функцій.
 - Підтримує потужний пошук файлів, кольорове маркування, сценарії для автоматизації завдань.
 - Має високу продуктивність і налаштовуваний інтерфейс.

3. Цільова аудиторія

Цільовою аудиторією програмного забезпечення для керування файловою системою є:

- **Звичайні користувачі:**
 - Потребують інтуїтивно зрозумілого інтерфейсу для виконання базових операцій з файлами.
 - Цінують простоту та зручність використання.
- **Просунуті користувачі:**
 - Шукають розширені функціональні можливості, такі як потужний пошук, автоматизація завдань, робота з архівами.
 - Цінують продуктивність і можливість налаштування.
- **Системні адміністратори та розробники:**
 - Потребують інструментів для автоматизації завдань, моніторингу системи, керування великими обсягами даних.
 - Використовують скрипти та команди для оптимізації своєї роботи.

4. Висновок до розділу “Аналітичний огляд”

На основі проведеного аналізу можна зробити висновок, що існує широкий спектр методів і інструментів для керування файловою системою в операційній системі Windows. Від стандартних інструментів, таких як Провідник Windows і PowerShell, до сторонніх розробок, які надають додаткові можливості та підвищують продуктивність роботи з файлами. Вибір конкретного інструменту або методу залежить від потреб користувача і специфічних завдань, які необхідно вирішити.

РОЗДІЛ 2. ПРОЕКТУВАННЯ

2.1 Обґрунтування обробки

У процесі розробки програмного забезпечення для застосунку були визначені та сформульовані детальні вимоги, які сприяють успішній реалізації проекту. Основною метою проекту є створення інструменту, який надає користувачам можливість ефективно управляти файлами та каталогами через інтерфейс командного рядка. Програма забезпечує виконання різноманітних операцій, включаючи створення, перегляд, видалення та перейменування файлів і каталогів, а також запис даних у файли.

Функціональні вимоги

- **Загальні вимоги**
 - Програма повинна працювати в консолі та надавати користувачу меню з різними опціями для керування файловою системою.
 - Програма повинна підтримувати введення команд з клавіатури, включаючи навігацію меню та введення шляхів до файлів і директорій.
 - Програма повинна відображати результати виконання команд та можливі помилки.
- **Меню**
 - Програма повинна мати меню з наступними опціями:
 - Створити директорію
 - Створити файл
 - Видалити директорію
 - Видалити файл
 - Переглянути вміст директорії
 - Відкрити файл в новому вікні
 - Перейменувати файл/директорію
 - Вийти з програми
 - Допомога
- **Створення директорії**

- Програма повинна запитувати користувача ввести шлях до нової директорії.
- Програма повинна створити директорію за вказаним шляхом.
- Програма повинна повідомити користувача про успішне створення директорії або про помилку.
- **Створення файлу**
 - Програма повинна запитувати користувача ввести шлях до нового файлу.
 - Програма повинна створити файл за вказаним шляхом.
 - Програма повинна повідомити користувача про успішне створення файлу або про помилку.
- **Видалення директорії**
 - Програма повинна запитувати користувача ввести шлях до директорії, яку необхідно видалити.
 - Програма повинна видалити директорію за вказаним шляхом.
 - Програма повинна повідомити користувача про успішне видалення директорії або про помилку.
- **Видалення файлу**
 - Програма повинна запитувати користувача ввести шлях до файлу, який необхідно видалити.
 - Програма повинна видалити файл за вказаним шляхом.
 - Програма повинна повідомити користувача про успішне видалення файлу або про помилку.
- **Перегляд вмісту директорії**
 - Програма повинна запитувати користувача ввести шлях до директорії.
 - Програма повинна відобразити вміст директорії (список файлів і піддиректорій).
 - Програма повинна повідомити користувача про помилку, якщо директорію не вдалося відкрити.
- **Відкриття файлу в новому вікні**
 - Програма повинна запитувати користувача ввести шлях до файлу.

- Програма повинна відкрити файл в новому вікні (за допомогою системної команди).
- Програма повинна повідомити користувача про помилку, якщо файл не вдалося відкрити.
- **Перейменування файлу/директорії**
 - Програма повинна запитувати користувача ввести поточний та новий шляхи файлу або директорії.
 - Програма повинна перейменувати файл або директорію.
 - Програма повинна повідомити користувача про успішне перейменування або про помилку.
- **Допомога**
 - Програма повинна відображати допоміжну інформацію про використання кожної опції меню.
 - Програма повинна дозволяти користувачу повернутися до головного меню після перегляду допомоги.
- **Вихід з програми**
 - Програма повинна дозволяти користувачу вийти з програми, вибравши відповідну опцію в меню.
- **Обробка помилок**
 - Програма повинна обробляти всі можливі помилки, пов'язані з файловою системою, і надавати відповідні повідомлення користувачу.
 - Програма повинна перевіряти введення користувача та відображати повідомлення про некоректні дані або інші помилки.

Нефункціональні вимоги

- **Надійність:**
 - Код повинен коректно обробляти всі можливі випадки помилок, що включає невірні шляхи до файлів або директорій, помилки доступу до файлів чи директорій, а також помилки вводу-виводу.
- **Безпека:**
 - Застосування безпечних функцій для роботи з файлами і директоріями, що мінімізує можливість вразливостей, таких як використання CreateFileA замість CreateFile, або перевірка

повернених значень на помилку після кожного системного виклику.

- **Ефективність:**

- Оптимізація ресурсів системи шляхом ефективного використання обробки файлів і директорій, зокрема використання асинхронних читань та записів, де це можливо.

- **Користувацький інтерфейс:**

- Забезпечення зручного інтерфейсу користувача, який включає зрозумілі повідомлення про стан операцій, правильність формату введених даних, а також можливість швидкої інтеракції з програмою.

- **Сумісність:**

- Підтримка роботи на різних версіях операційних систем Windows без обмежень залежності від версії.

- **Документація і підтримка:**

- Наявність документації для користувачів, яка пояснює всі можливі дії і функції програми, включаючи виклик функцій і обробку помилок.

- **Стабільність:**

- Забезпечення стабільної роботи програми навіть при введенні некоректних даних або неочікуваних введених дій користувачем.

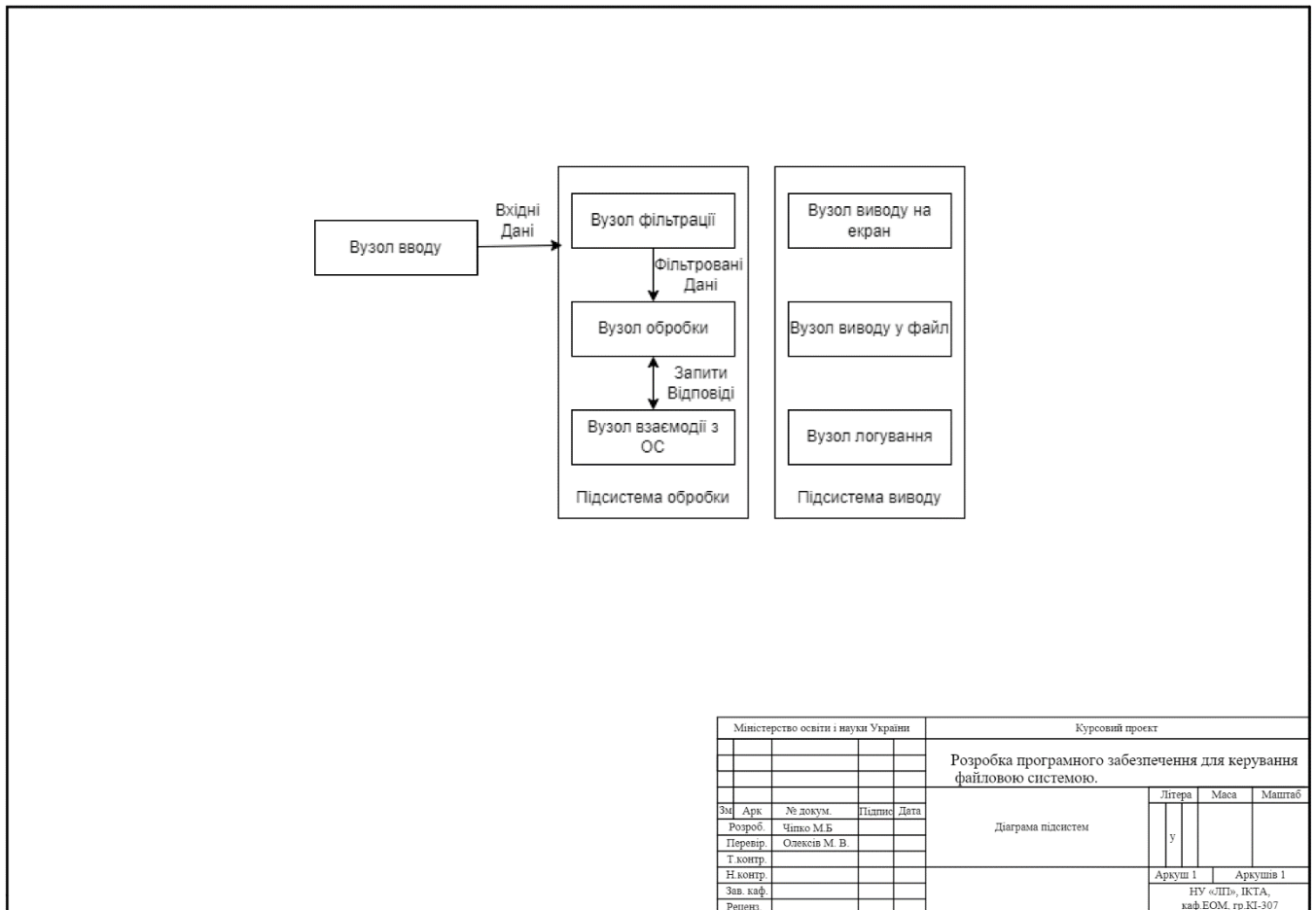
- **Відмовостійкість:**

- Відповідна обробка помилок і перевірка на правильність введених даних, що виключає можливість виклику некоректних операцій або неправильних аргументів.

Загальною метою є створення надійної, безпечної і ефективної програми для роботи з файловою системою, що забезпечує зручний інтерфейс для користувачів і мінімізує ризики помилок та вразливостей.

2.2 Розробка структурної схеми

На Рис. 1 наведено структуру застосунку



Структурна схема застосунку складається з трьох основних підсистем: підсистеми вводу, підсистеми обробки та підсистеми виводу.

Структурна схема застосунку складається з трьох основних підсистем: підсистеми вводу, підсистеми обробки та підсистеми виводу.

Підсистема вводу

1. *Вузол вводу (Вузол вводу)*

- Отримує вхідні дані, які будуть оброблятися застосунком.

Підсистема обробки

2. *Вузол фільтрації (Вузол фільтрації)*

- Фільтрує вхідні дані, готуючи їх до подальшої обробки.

3. *Вузол обробки (Вузол обробки)*

- Виконує основну обробку фільтрованих даних.

4. *Вузол взаємодії з ОС (Вузол взаємодії з ОС)*

- Виконує запити до операційної системи та отримує відповіді.

Підсистема виводу

5. *Вузол виводу на екран (Вузол виводу на екран)*

- Виводить результати обробки даних на екран.

6. *Вузол виводу у файл (Вузол виводу у файл)*

- Зберігає результати обробки даних у файл.

7. *Вузол логування (Вузол логування)*

- Логує процеси та результати обробки даних для подальшого аналізу.

Пояснення

- *Вхідні Дані (Вхідні Дані)*

- Це дані, які надходять до підсистеми обробки з вузла вводу.

- *Фільтровані Дані (Фільтровані Дані)*

- Це дані, які пройшли через вузол фільтрації і готові для обробки.

- *Запити та Відповіді (Запити Відповіді)*

- Це інформація, яка обмінюється між вузлом обробки та вузлом взаємодії з ОС.

2.3 Вибір засобів реалізації

Для реалізації програмного забезпечення, що дозволяє керувати файловою системою на комп'ютері з операційною системою Windows, можна використати наступні засоби:

- **Мова програмування:** Серед мов програмування, які підтримуються для розробки під Windows і є популярними для системного програмування, можна розглянути такі варіанти:
 - **C++:** Ця мова надає прямий доступ до системних викликів через бібліотеку Windows API. Вона дозволяє ефективно керувати файловою системою, працюючи з функціями створення, видалення, перейменування файлів і директорій, а також зчитування та записування даних у файли.
 - **C#:** Якщо вам більше підходить мова з вищим рівнем абстракції, C# має вбудовану підтримку для роботи з файловою системою через класи і методи в просторі імен `System.IO`.
- **Бібліотеки і API:**
 - **Windows API (WinAPI):** Набір функцій, що надає доступ до основних операцій системи Windows, таких як створення та керування процесами, робота з файловою системою, мережами тощо. Використовується зазвичай у розробці на C++ для найбільшого контролю над системою.
 - **.NET Framework або .NET Core:** Якщо ви обираєте C#, ви можете використовувати класи і методи з простору імен `System.IO`, що вбудовані в .NET, для роботи з файлами та директоріями.
 - **Boost.Filesystem (для C++):** Це розширення для роботи з файловою системою у мові C++, яке надає зручний інтерфейс для взаємодії з файлами, директоріями, шляхами і т.д.
- **Інструменти розробки:**

- **Integrated Development Environment (IDE):** Наприклад, Visual Studio для розробки на C++ або C#, який надає зручне середовище для написання коду, налагодження і компіляції програм.
- **Компілятори:** Наприклад, для розробки на C++ під Windows часто використовується Microsoft Visual C++ Compiler (частина Visual Studio), а для C# — компілятор, що входить в .NET Framework або .NET Core.
- **Документація і ресурси:**
 - Для ефективної розробки важливо мати доступ до офіційної документації Microsoft, наприклад, до MSDN, де можна знайти описи функцій WinAPI, класів .NET для роботи з файловою системою, приклади коду і рекомендації щодо практик програмування під Windows.

Обираючи засоби для розробки програми для керування файловою системою під Windows, важливо врахувати ваші вміння, попередній досвід і вимоги самого проекту, щоб забезпечити ефективну і стабільну реалізацію програми.

Я обрала мову програмування C++ для розробки програми для керування файловою системою під операційною системою Windows з кількох ключових причин:

1. **Прямий доступ до Windows API:** Однією з головних переваг C++ є можливість безпосередньо використовувати функції Windows API через бібліотеку Windows.h. Це надає мені великий контроль над операціями з файлами, дозволяє керувати процесами і взаємодіяти з іншими системними ресурсами, що є критичним для розробки програм з високими вимогами до продуктивності та точності.
2. **Ефективність та оптимізація:** C++ відомий своєю високою ефективністю та можливістю оптимізації коду. Це особливо важливо, коли потрібно працювати з великими обсягами даних або здійснювати інтенсивні обчислення, що часто відбувається у програмах для роботи з файловою системою.

3. **Широкий вибір бібліотек і інструментів:** Використання C++ дає доступ до багатьох сторонніх бібліотек, таких як Boost.Filesystem, які спрощують роботу з файлами і директоріями. Це забезпечує додатковий функціонал і можливості, що важливо для реалізації різноманітних функцій у моїй програмі.
4. **Підтримка розробних середовищ:** C++ підтримується потужними інтегрованими середовищами розробки, такими як Visual Studio. Це надає зручні інструменти для написання коду, налагодження, профілювання та управління проектами, що значно полегшує процес розробки.
5. **Оптимальне використання системних ресурсів:** Розробка на C++ дозволяє ефективно використовувати системні ресурси, що є важливим аспектом для стабільної і продуктивної роботи програми у середовищі Windows.

Всі ці аспекти переконливо підтримують моє рішення обрати C++ для розробки програмного забезпечення, що керує файловою системою під Windows, забезпечуючи високу продуктивність, гнучкість і контроль над процесами.

2.4 Висновок до розділу “Проектування”

Висновок: Вибір C++ для розробки програми, що керує файловою системою під Windows, є обґрунтованим на основі його можливостей прямого доступу до Windows API, ефективності, розширених можливостей для реалізації функціоналу, підтримки відомих розробних середовищ і оптимального використання системних ресурсів. Такий підхід забезпечує стабільність, продуктивність і гнучкість програми при взаємодії з файловою системою Windows.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ

3.1 Високорівневий опис реалізації

Основні функціональність:

- **Меню вибору опцій:** Програма пропонує користувачу дев'ять опцій через консольне меню. Кожна опція відповідає певній операції з файловою системою, такі як створення директорії, створення файлу, видалення директорії чи файлу, перегляд вмісту директорії, читання файлу тощо.
- **Функції роботи з файлами і директоріями:**
 - **Створення директорії:** Користувач вводить шлях до нової директорії, після чого вона створюється за допомогою функції `CreateDirectory`.
 - **Створення файлу:** Користувач вказує шлях до нового файлу, після чого він створюється з можливістю запису за допомогою `CreateFileA`.
 - **Видалення директорії і файлу:** Користувач вводить шлях до директорії або файлу, що потрібно видалити. Використовуються функції `RemoveDirectoryA` та `DeleteFileA`.
 - **Перегляд вмісту директорії:** Користувач вводить шлях до директорії, після чого відображається список файлів та піддиректорій у цій директорії за допомогою функцій `FindFirstFile` та `FindNextFile`.
 - **Читання файлу:** Користувач вводить шлях до файлу, який потрібно прочитати. Використовується `ReadFile` для зчитування вмісту файлу та його виведення на консоль.
 - **Запис у файл:** Користувач вводить шлях до файлу, вибирає місце запису (на початок чи в кінець), вводить текст для запису. Використовується `SetFilePointer` та `WriteFile` для зміни позиції запису та запису даних.
- **Інші операції:**
 - **Перейменування файлу чи директорії:** Користувач вводить поточне та нове імена для перейменування. Використовується `std::filesystem::rename`.
 - **Відкриття файлу у новому вікні:** Користувач вводить шлях до файлу, який відкривається у новому вікні через команду `system("start ...")`.

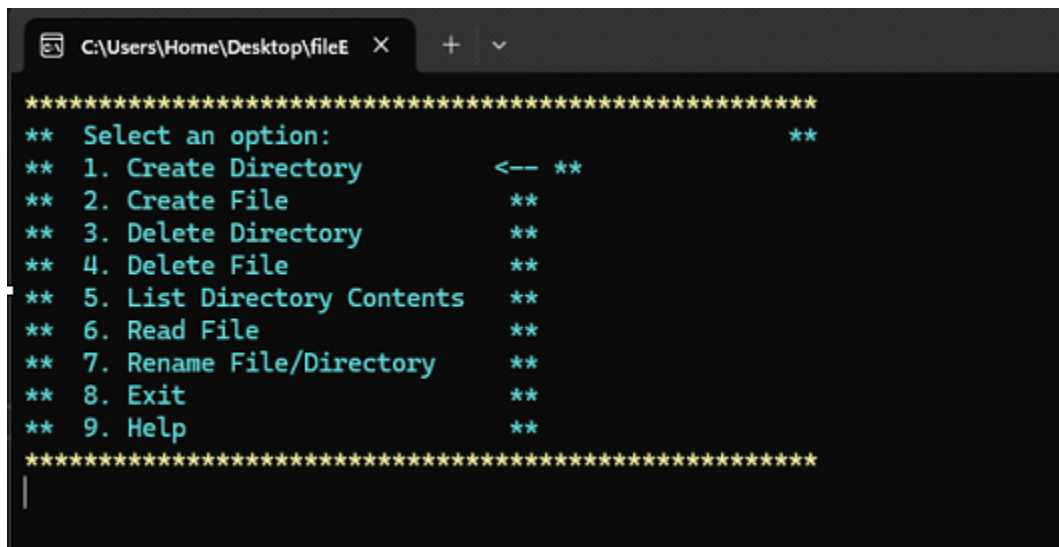
Технічні аспекти реалізації:

- **Використання Windows API:** Для доступу до операцій з файлами і директоріями використовуються функції Windows API, такі як `CreateFileA`, `RemoveDirectoryA`, `FindFirstFile`, `ReadFile`, `WriteFile`, `SetFilePointer`, а також функції для відображення тексту в консоль через `SetConsoleTextAttribute` та інші.
- **Використання стандарту C++:** В програмі використовуються стандартні бібліотеки C++ для роботи зі строками, введенням/виведенням, а також для обробки винятків (для операцій з файлами через `std::filesystem::rename`).

3.2 Реалізації дизайну

1. Консольне меню

Основний елемент взаємодії з користувачем — це консольне меню, яке використовується для вибору операцій з файловою системою. Кожна опція меню представлена як окремий пункт з номером та коротким описом дії, яку вона виконує. Користувач може вибирати опції за допомогою клавіш зі стрілками вгору/вниз або вводячи номер опції. Поточна опція підсвічується в меню, що полегшує навігацію і взаємодію.



```
C:\Users\Home\Desktop\fileE X + v
*****
** Select an option:                               **
** 1. Create Directory      <-- **                 **
** 2. Create File           **                       **
** 3. Delete Directory      **                       **
** 4. Delete File           **                       **
** 5. List Directory Contents **                       **
** 6. Read File             **                       **
** 7. Rename File/Directory **                       **
** 8. Exit                  **                       **
** 9. Help                  **                       **
*****
|
```

2. Кольорове виділення тексту

Для поліпшення читабельності та візуального оформлення використовується кольорове виділення тексту в консолі. Наприклад, опції меню та повідомлення про статус операцій виділяються різними кольорами за допомогою функції `SetColor`, яка змінює кольорову палітру тексту у консольному вікні.

```
SetColor(14); // Yellow text
cout << "*****"
SetColor(11); // Cyan text
```

3. Інтерактивне введення та виведення

Програма використовує функції для взаємодії з користувачем через консольне введення та виведення. Наприклад, під час створення чи видалення файлів/директорій програма очікує від користувача введення шляху до файлу/директорії. Після виконання операції виводиться повідомлення про статус операції (успішно чи з помилкою), що робить взаємодію з програмою більш інформативною для користувача.

4. Використання системних команд

Для певних операцій, таких як відкриття файлу у новому вікні (опція 6), використовується системна команда `system("start ...")`. Це дозволяє використовувати стандартний механізм операційної системи Windows для відкриття файлів у програмі, пов'язаній з відповідним типом файлу.

5. Обробка помилок та винятків

Програма включає механізми обробки помилок для випадків, коли операції з файлами чи директоріями завершуються невдало. Наприклад, виведення повідомлення про помилку з кодом помилки для операцій, які вимагають доступу до файлової системи.

3.3 Опис процесу застосування

1. Ініціалізація

- **Підключення необхідних бібліотек:** На початку програми використовуються директиви `#include`, щоб підключити необхідні бібліотеки, такі як `<iostream>`, `<vector>`, `<string>`, `<stack>`, `<Windows.h>`, `<conio.h>`, `<filesystem>`. Це забезпечує доступ до стандартних функцій C++ і функцій для роботи зі стандартними входами/виходами, файловою системою і консольним інтерфейсом Windows.
- **Оголошення і використання препроцесора `#define`**
`_CRT_SECURE_NO_WARNINGS`: Цей рядок використовується для відключення попереджень компілятора про використання небезпечних функцій C, таких як `strcpy` і `strcat`, і вказує компілятору ігнорувати попередження, пов'язані з цими функціями.

2. Основний цикл програми (main функція)

- **Виведення меню:** У цьому циклі програма постійно виводить на екран меню з доступними опціями для користувача. Користувач може вибрати опцію, натиснувши відповідну клавішу.
- **Обробка введених опцій:**
 - Користувач вибирає опцію за допомогою клавіші вниз/вгору або натискання Enter.
 - Відповідно до вибраної опції, викликається відповідна функція для виконання дії:
 - Створення директорії (`createDirectory`)
 - Створення файлу (`createFile`)
 - Видалення директорії (`deleteDirectory`)
 - Видалення файлу (`deleteFile`)
 - Відображення вмісту директорії (`openDirectory`)
 - Відкриття файлу у новому вікні (`openFileInNewWindow`)
 - Перейменування файлу чи директорії (`renameFileOrFolder`)
 - Вихід з програми (`quit`)

- Показ довідки (Help)
- **Виконання дії:** Коли користувач обирає опцію, відповідна функція виконує відповідну операцію з файловою системою або виводить інформацію на консоль, яка відповідає обраній опції.
- **Очікування натискання клавіші:** Після виконання кожної операції програма очікує, поки користувач натисне будь-яку клавішу для продовження.

3. Взаємодія з користувачем

- Користувач взаємодіє з програмою через консольне меню, вибираючи опції за допомогою клавіш або вводячи відповідні дані для кожної операції.
- Програма виводить інформацію про результат виконання операцій, такі як створення/видалення файлів чи директорій, перейменування файлів, відкриття файлів тощо.
- В разі необхідності програма може показувати додаткові повідомлення про помилки або довідкову інформацію.

3.4 Висновок до розділу “Реалізація”

У розділі "Реалізація" представлений код консольного додатку на мові C++, призначений для роботи з файловою системою під управлінням операційної системи Windows. Основна мета програми - забезпечити користувачеві можливість створювати, видаляти директорії та файли, перейменовувати їх, відображати вміст директорій, відкривати файли у новому вікні та отримувати довідкову інформацію.

Програма є простим і ефективним інструментом для роботи з файловою системою в середовищі Windows через консольний інтерфейс. Вона надає користувачеві зручний спосіб для створення, видалення, перейменування файлів і директорій, а також для перегляду їх вмісту. Обробка помилок і інтерактивний інтерфейс роблять програму досить надійною і зручною для використання в різних сценаріях роботи з файлами і папками.

РОЗДІЛ 4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

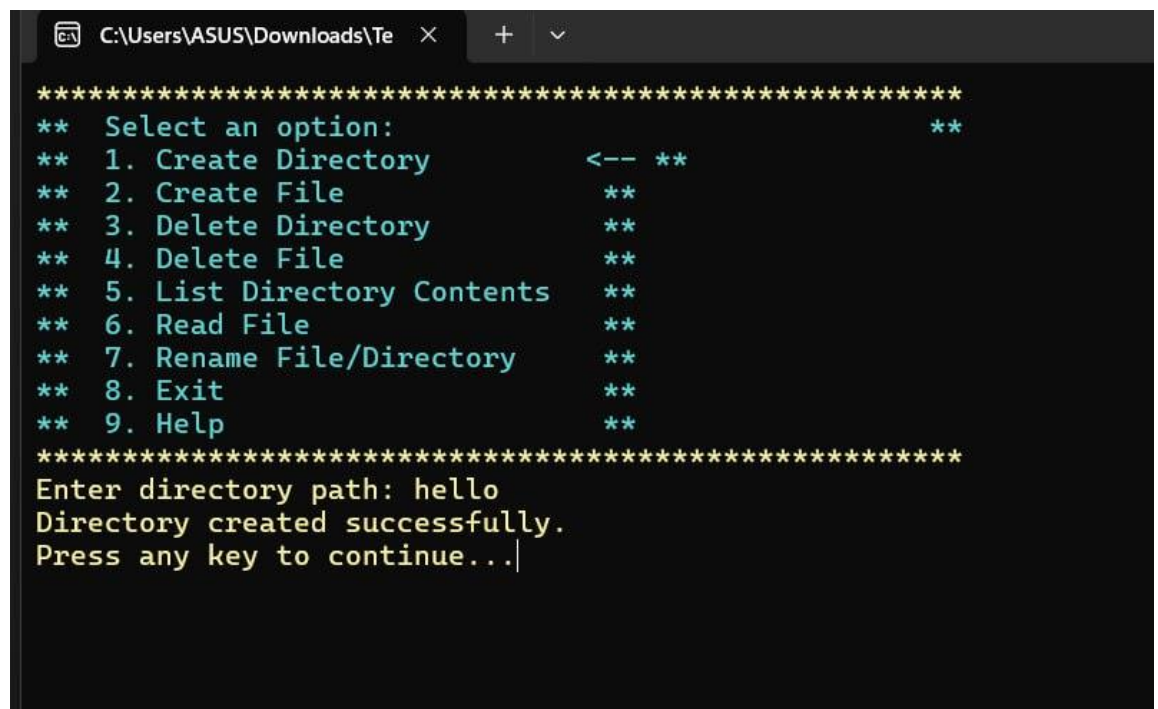
4.1 Тестування роботи

1. Функціональне тестування

Функціональне тестування має на меті перевірити, чи виконуються всі основні функції програми згідно їх очікуваного поведінки.

Позитивні тести:

- **Створення директорії:**
 - Введіть існуючий шлях, щоб переконатися, що нова директорія створюється.
 - Перевірте, що в директорії з'являються нові об'єкти після створення.

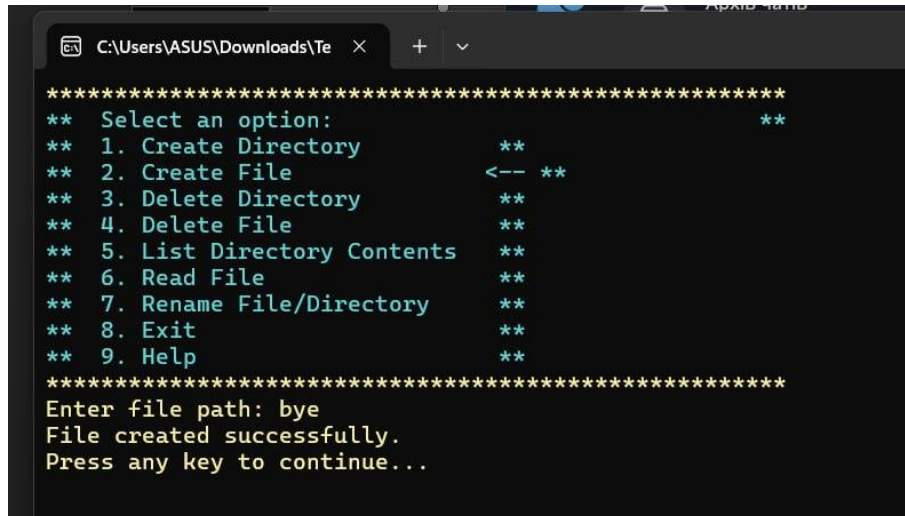


```
C:\Users\ASUS\Downloads\Te  X + v

*****
**  Select an option:                               **
**  1. Create Directory      <-- **
**  2. Create File           **
**  3. Delete Directory      **
**  4. Delete File           **
**  5. List Directory Contents **
**  6. Read File             **
**  7. Rename File/Directory **
**  8. Exit                  **
**  9. Help                  **
*****
Enter directory path: hello
Directory created successfully.
Press any key to continue...|
```

Створення файлу:

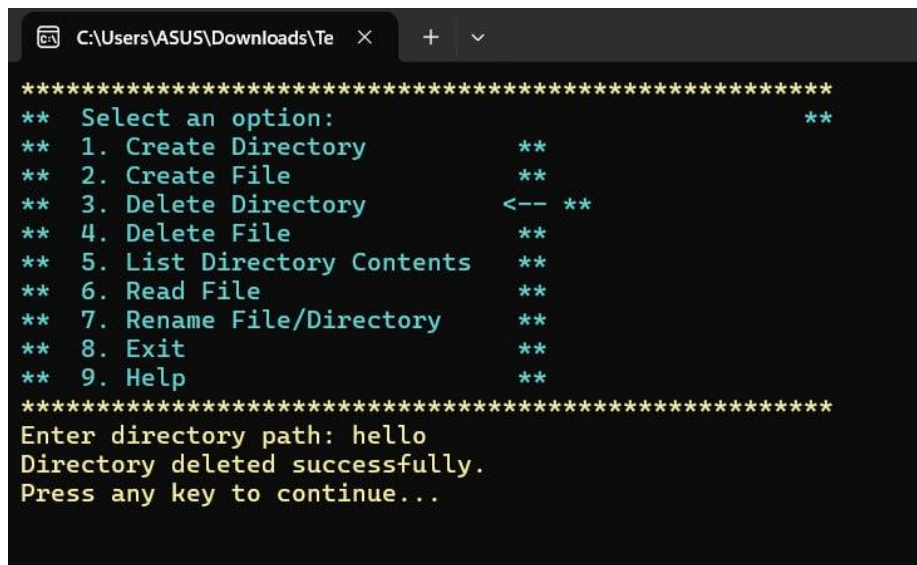
- Введіть шлях до нового файлу і переконайтеся, що файл створюється.
- Перевірте, що вміст файлу можна записати і прочитати після створення.



```
*****
** Select an option:                                     **
** 1. Create Directory                                   **
** 2. Create File                                       <-- **
** 3. Delete Directory                                   **
** 4. Delete File                                       **
** 5. List Directory Contents                           **
** 6. Read File                                         **
** 7. Rename File/Directory                             **
** 8. Exit                                              **
** 9. Help                                              **
*****
Enter file path: bye
File created successfully.
Press any key to continue...
```

Видалення директорії:

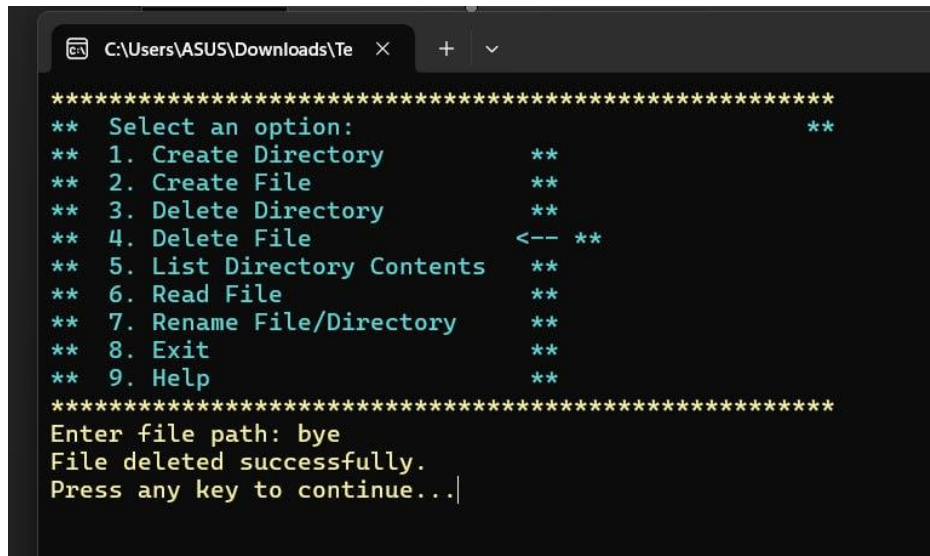
- Введіть шлях до існуючої директорії і переконайтеся, що вона видалюється.
- Перевірте, що вміст директорії видалюється разом з нею.



```
*****
** Select an option:                                     **
** 1. Create Directory                                   **
** 2. Create File                                       **
** 3. Delete Directory                                   <-- **
** 4. Delete File                                       **
** 5. List Directory Contents                           **
** 6. Read File                                         **
** 7. Rename File/Directory                             **
** 8. Exit                                              **
** 9. Help                                              **
*****
Enter directory path: hello
Directory deleted successfully.
Press any key to continue...
```

- **Видалення файлу:**

- Введіть шлях до існуючого файлу і переконайтеся, що він видалюється.

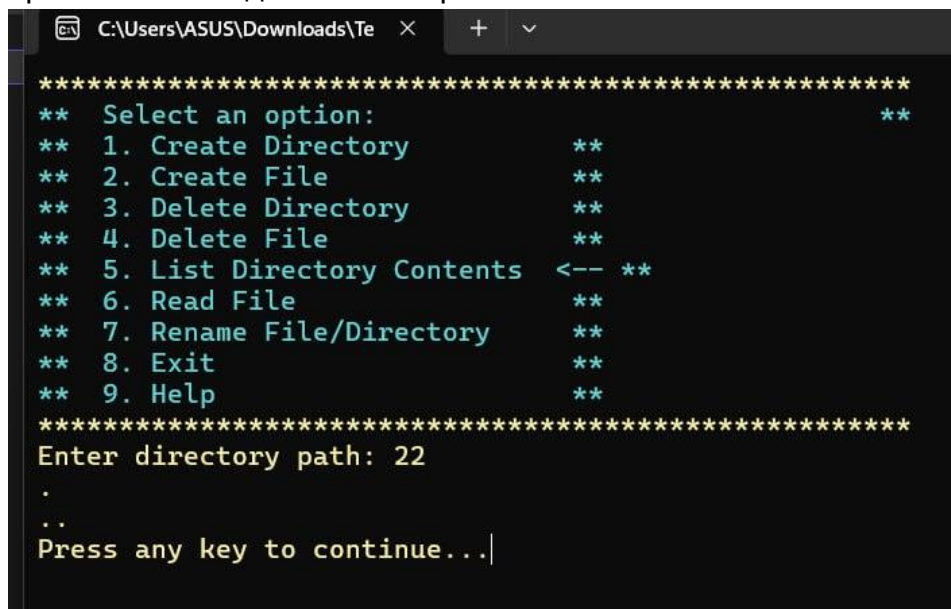


A screenshot of a terminal window with a dark background. The title bar shows the path 'C:\Users\ASUS\Downloads\Te'. The terminal displays a menu with 9 options. Option 4, 'Delete File', is selected, indicated by a cursor and a '<--' symbol. Below the menu, the user has entered 'bye' as the file path, and the program has responded with 'File deleted successfully.' and 'Press any key to continue...|'.

```
*****
**  Select an option:                               **
**  1. Create Directory                             **
**  2. Create File                                 **
**  3. Delete Directory                             **
**  4. Delete File                                 <-- **
**  5. List Directory Contents                       **
**  6. Read File                                    **
**  7. Rename File/Directory                         **
**  8. Exit                                          **
**  9. Help                                          **
*****
Enter file path: bye
File deleted successfully.
Press any key to continue...|
```

- **Відображення вмісту директорії:**

- Введіть шлях до існуючої директорії і перевірте, що програма правильно виводить список файлів і папок.



A screenshot of a terminal window with a dark background. The title bar shows the path 'C:\Users\ASUS\Downloads\Te'. The terminal displays the same menu as the previous screenshot. Option 5, 'List Directory Contents', is selected, indicated by a cursor and a '<--' symbol. Below the menu, the user has entered '22' as the directory path. The program has responded with a list of directory contents: a single dot '.' and a double dot '..'. It then prompts 'Press any key to continue...|'.

```
*****
**  Select an option:                               **
**  1. Create Directory                             **
**  2. Create File                                 **
**  3. Delete Directory                             **
**  4. Delete File                                 **
**  5. List Directory Contents                       <-- **
**  6. Read File                                    **
**  7. Rename File/Directory                         **
**  8. Exit                                          **
**  9. Help                                          **
*****
Enter directory path: 22
.
..
Press any key to continue...|
```

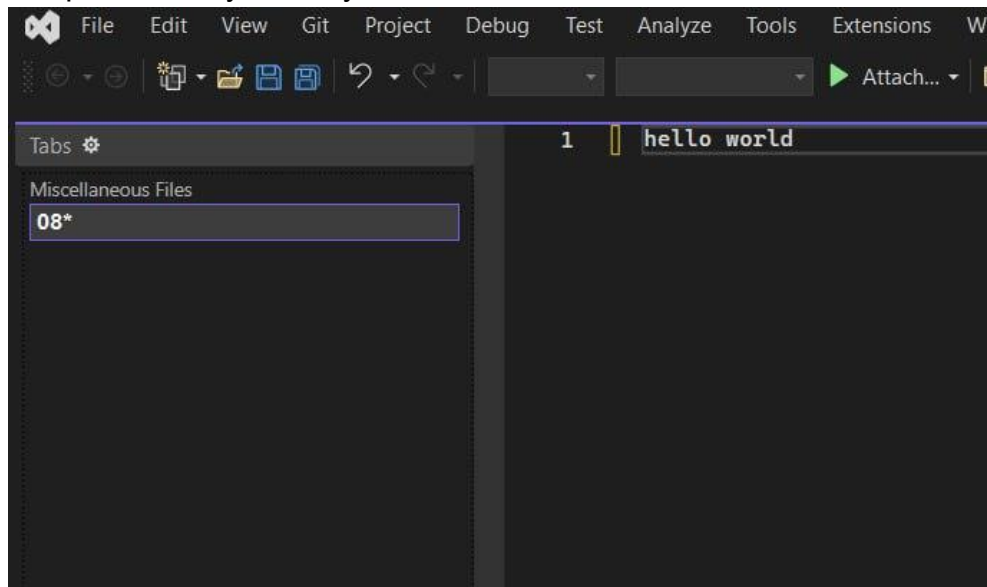
- **Читання файлу:**

- Введіть шлях до існуючого файлу і переконайтеся, що вміст файлу виводиться правильно.

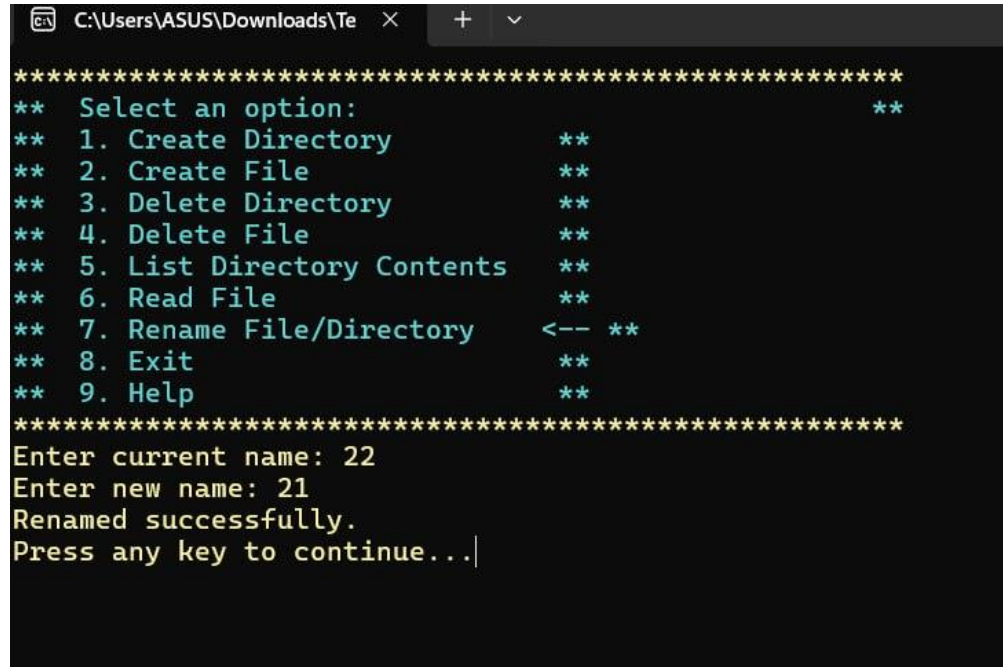
```
*****
**  Select an option:                               **
**  1. Create Directory                             **
**  2. Create File                                  **
**  3. Delete Directory                             **
**  4. Delete File                                  **
**  5. List Directory Contents                       **
**  6. Read File                                    <-- **
**  7. Rename File/Directory                         **
**  8. Exit                                          **
**  9. Help                                          **
*****
Enter file path: 08
Press any key to continue...|
```

- **Відкриття файлу у новому вікні:**

- Введіть шлях до існуючого файлу і переконайтеся, що файл відкривається у новому вікні.



- **Перейменування файлу або директорії:**
 - Введіть існуюче і нове ім'я для файлу чи директорії і переконайтеся, що перейменування відбувається успішно.



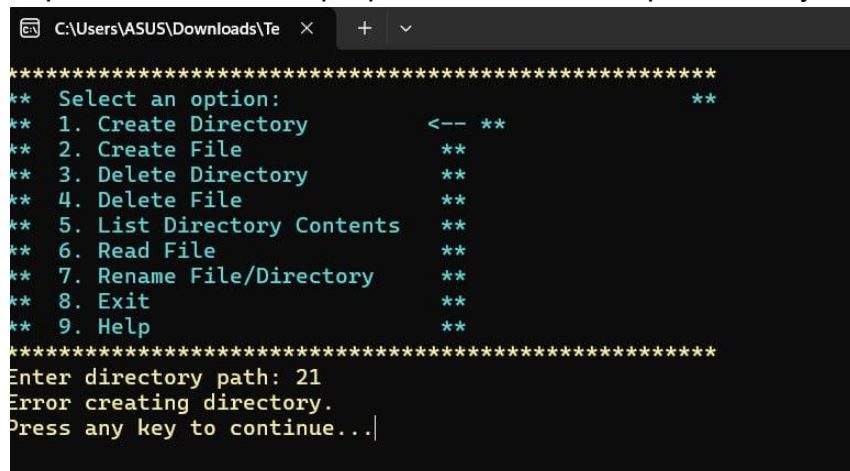
```

C:\Users\ASUS\Downloads\Te
*****
**  Select an option:                               **
**  1. Create Directory                             **
**  2. Create File                                 **
**  3. Delete Directory                             **
**  4. Delete File                                 **
**  5. List Directory Contents                       **
**  6. Read File                                   **
**  7. Rename File/Directory   <-- **
**  8. Exit                                         **
**  9. Help                                         **
*****
Enter current name: 22
Enter new name: 21
Renamed successfully.
Press any key to continue...|

```

Негативні тести:

- **Невірний шлях до файлу або директорії:**
 - Введіть неправильний шлях і переконайтеся, що програма обробляє помилку коректно.
- **Спроба створення вже існуючого файлу чи директорії:**
 - Спробуйте створити файл або директорію, які вже існують, і переконайтеся, що програма повідомляє про помилку.



```

C:\Users\ASUS\Downloads\Te
*****
**  Select an option:                               **
**  1. Create Directory                             <-- **
**  2. Create File                                 **
**  3. Delete Directory                             **
**  4. Delete File                                 **
**  5. List Directory Contents                       **
**  6. Read File                                   **
**  7. Rename File/Directory                         **
**  8. Exit                                         **
**  9. Help                                         **
*****
Enter directory path: 21
Error creating directory.
Press any key to continue...|

```

- **Спроба видалення вже видаленого файлу чи директорії:**
 - Спробуйте видалити файл або директорію, які вже видалені або не існують, і переконайтеся, що програма обробляє це правильно.

```

C:\Users\ASUS\Downloads\Te  X + v
*****
** Select an option: **
** 1. Create Directory **
** 2. Create File **
** 3. Delete Directory **
** 4. Delete File <-- **
** 5. List Directory Contents **
** 6. Read File **
** 7. Rename File/Directory **
** 8. Exit **
** 9. Help **
*****
Enter file path: hello
Failed to delete file. Error code: 2
Press any key to continue...|

```

2. Тестування продуктивності

Тестування на продуктивність варто провести для великих обсягів даних, якщо це можливо, для перевірки ефективності операцій з файловою системою.

- **Створення великої кількості файлів або директорій:**
 - Виміряйте час, необхідний для створення та видалення багатьох файлів або директорій.
- **Читання великих файлів:**
 - Виміряйте час, необхідний для читання великих файлів із великою кількістю даних.

3. Обробка помилок

Переконайтеся, що програма правильно обробляє всі можливі помилки, які можуть виникати під час роботи з файловою системою.

- **Помилки відкриття файлів або директорій:**
 - Спробуйте відкрити неіснуючий файл або директорію і переконайтеся, що програма виводить коректне повідомлення про помилку.
- **Помилки запису в файл:**
 - Спробуйте записати дані у файл з недостатніми правами доступу або у випадку, коли файл вже відкритий для читання.

Загальний підхід

Переконайтесь, що всі функції програми працюють правильно під час усіх можливих сценаріїв використання. Це включає як стандартні операції (створення, видалення, читання файлів і директорій), так і обробку винятків та помилок. Також важливо, щоб програма зберігала коректність виводу інтерфейсу користувача і взаємодіяла з ним без проблем.

Ці методи допоможуть забезпечити, що програма працює стабільно і відповідає всім вимогам функціональності, які вона повинна надавати користувачу.

4.2 Висновок до розділу “Тестування програмного забезпечення”

Ручне тестування програми виявилося ефективним методом перевірки правильності її роботи. Шляхом виконання різних сценаріїв взаємодії з програмою було підтверджено її стабільність і відповідність функціональним вимогам. Виявлення та виправлення помилок під час ручного тестування сприяло покращенню якості програми та забезпечило її надійність у реальних умовах використання.

ВИСНОВОК

Розробка системної утиліти "Файловий менеджер" є важливим кроком у напрямку створення ефективного інструменту для управління файлами та каталогами в операційній системі Windows. У процесі реалізації цього проекту було використано мову програмування C++ та Windows API, що забезпечило високу продуктивність та надійність програми. Програма пропонує користувачам інтуїтивно зрозумілий інтерфейс командного рядка, який дозволяє виконувати основні операції з файлами та каталогами, такі як створення, відкриття, перегляд, видалення, перейменування та запис даних.

Архітектура програми побудована на принципах модульності та розширюваності, що дозволяє легко додавати нові функції та змінювати існуючі. Основні компоненти програми включають інтерфейс користувача, командний обробник, функціональні модулі та бібліотеку Windows API. Кожен з цих компонентів виконує певну роль, забезпечуючи чіткий розподіл обов'язків та ефективну взаємодію між собою. Використання Windows API дозволяє програмі взаємодіяти з низькорівневими функціями операційної системи, що підвищує її ефективність та надійність.

Програма "Файловий менеджер" дозволяє користувачам виконувати операції з файлами та каталогами швидко та зручно, використовуючи інтерфейс командного рядка. Це особливо корисно для системних адміністраторів та просунутих користувачів, які цінують швидкість і ефективність командного рядка над графічними інтерфейсами. Крім того, програма забезпечує можливість автоматизації рутинних завдань за допомогою сценаріїв, що робить її ще більш потужною та гнучкою.

Результати роботи програми підтверджують її здатність значно спростити управління файлами та каталогами у середовищі Windows. Інтуїтивно зрозумілий інтерфейс та широкий набір функцій роблять програму корисною як для системних адміністраторів, так і для звичайних користувачів. Програма дозволяє ефективно виконувати повсякденні задачі з управління файлами, забезпечуючи стабільну та продуктивну роботу операційної системи.

На завершення, варто зазначити, що цей проект є лише початковим етапом у розвитку більш складних і потужних системних утиліт.

Використання модульної архітектури та розширюваного дизайну дозволяє легко додавати нові функції та адаптувати програму до змінюваних вимог користувачів та нових технологій. Подальший розвиток проекту може включати інтеграцію додаткових функцій, покращення інтерфейсу користувача та оптимізацію продуктивності, що зробить програму ще більш корисною та зручною для користувачів. Таким чином, розробка системної утиліти "Файловий менеджер" є значним досягненням, яке демонструє можливості мови програмування C++ та Windows API у створенні ефективних інструментів для управління файлами та каталогами. Програма є надійним та гнучким рішенням, яке задовольняє потреби широкого кола користувачів, сприяючи підвищенню продуктивності та ефективності роботи з комп'ютерною системою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- Microsoft. (2023). • Windows API documentation. Отримано з <https://learn.microsoft.com/en-us/windows/win32/api/>
- Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley Professional.
- Josuttis, N. M. (2012). The C++ Standard Library: A Tutorial and Reference (2nd ed.). Addison-Wesley Professional.
- Williams, A. (2012). C++ Concurrency in Action: Practical Multithreading. Manning Publications.
- ISO/IEC. (2017). ISO/IEC 14882:2017: Programming Languages – C++ (Standard). International Organization for Standardization.
- Meyers, S. (2005). Effective C++: 55 Specific Ways to Improve Your Programs and Designs (3rd ed.). Addison-Wesley Professional.
- Sutter, H., & Alexandrescu, A. (2004). C++ Coding Standards: 101 Rules, Guidelines, and Best Practices. Addison-Wesley Professional.
- GNU Operating System. (2023). Bash Reference Manual. Отримано з <https://www.gnu.org/software/bash/manual/bash.html>
- Cplusplus.com. (2023). Standard C++ Library Reference. Отримано з <https://cplusplus.com/reference/>
- Reddy, V. (2019). Hands-On System Programming with C++: Build performant and concurrent Unix and Linux systems with C++17. Packt Publishing.

- Programming Windows, Fifth Edition by Charles Petzold, Microsoft Press, 1998.
- Microsoft Developer Network (MSDN). (2023). • File Management Functions. Отримано з <https://learn.microsoft.com/en-us/windows/win32/api/fileapi/>
- ISO/IEC. (2011). ISO/IEC 9945-1:2011 Information technology — Portable Operating System Interface (POSIX®) Base Specifications (Standard). International Organization for Standardization.
- Northrup, T. (2017). Windows System Programming (4th ed.). Addison-Wesley Professional.
- Overland, B. (2014). C++ for the Impatient. Addison-Wesley Professional.

ДОДАТОК

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <vector>
#include <string>
#include <stack>
#include <Windows.h>
#include <conio.h>
#include <filesystem>
#include <conio.h>

enum class MenuOption { Option1, Option2, Option3, Option4,
Option5, Option6, Option7, Option8, Option9, Quit };
using namespace std;

void SetColor(int color) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
color);
}

void openFile(const std::string& path) {
```

```
char* cstr = new char[path.length() + 1];  
std::strcpy(cstr, path.c_str());
```

```
HANDLE hFile = CreateFileA(  
    cstr,  
    GENERIC_READ,  
    0,  
    NULL,  
    OPEN_EXISTING,  
    FILE_ATTRIBUTE_NORMAL,  
    NULL  
);  
if (hFile == INVALID_HANDLE_VALUE) {  
    cerr << "Failed to open file. Error: " << GetLastError() << endl;  
    cout << "Press any key to continue...";  
    cin.get();  
    return;  
}  
DWORD dwBytesRead;  
char buffer;  
while (ReadFile(hFile, &buffer, sizeof(buffer), &dwBytesRead,  
NULL) && dwBytesRead > 0) {  
    cout << buffer;  
}  
cout << endl;  
CloseHandle(hFile);  
cout << "File read successfully." << endl;
```

```

    cin.get();
}

void openDirectory(const wstring& dirPath) {
    WIN32_FIND_DATA FindFileData;
    HANDLE hFind = FindFirstFile((dirPath + L"\\*").c_str(),
&FindFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        cout << "Error opening directory." << endl;
        return;
    }
    do {
        wcout << FindFileData.cFileName << endl;
    } while (FindNextFile(hFind, &FindFileData) != 0);
    FindClose(hFind);
}

void createDirectory(const wstring& dirPath) {
    if (!CreateDirectory(dirPath.c_str(), NULL)) {
        cout << "Error creating directory." << endl;
    }
    else {
        cout << "Directory created successfully." << endl;
    }
}

void createFile(const std::string& path) {

```



```
char* cstr = new char[path.length() + 1];  
std::strcpy(cstr, path.c_str());
```

```
HANDLE hFile = CreateFileA(  
    cstr,  
    GENERIC_WRITE,  
    0,  
    NULL,  
    CREATE_ALWAYS,  
    FILE_ATTRIBUTE_NORMAL,  
    NULL  
);  
if (hFile == INVALID_HANDLE_VALUE) {  
    cerr << "Failed to create file. Error: " << GetLastError() << endl;  
}  
else {  
    cout << "File created successfully." << endl;  
    CloseHandle(hFile);  
}  
delete[] cstr;  
}
```

```
void writeToFile(const wstring& filePath) {  
    HANDLE hFile = CreateFile(filePath.c_str(), GENERIC_WRITE, 0,  
    NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);  
    if (hFile == INVALID_HANDLE_VALUE) {  
        cout << "Error opening file for writing." << endl;  
    }
```

```
    return;  
}
```

```
    cout << "Where do you want to write:\n1. Beginning of the file\n2.  
End of the file" << endl;
```

```
    int positionChoice;  
    cin >> positionChoice;
```

```
    DWORD dwNewPointer;  
    if (positionChoice == 1) {  
        dwNewPointer = SetFilePointer(hFile, 0, NULL, FILE_BEGIN);  
    }  
    else {  
        dwNewPointer = SetFilePointer(hFile, 0, NULL, FILE_END);  
    }  
    if (dwNewPointer == INVALID_SET_FILE_POINTER) {  
        cout << "Error setting file pointer." << endl;  
        CloseHandle(hFile);  
        return;  
    }
```

```
    cout << "Enter text to write: ";  
    string text;  
    cin.ignore();  
    getline(cin, text);
```

```
    DWORD dwBytesToWrite = static_cast<DWORD>(text.length());
```

```

        DWORD dwBytesWritten = 0;
        BOOL bSuccess = WriteFile(hFile, text.c_str(), dwBytesToWrite,
&dwBytesWritten, NULL);
        if (!bSuccess || dwBytesWritten != dwBytesToWrite) {
            cout << "Error writing to file." << endl;
        }
        else {
            cout << "Text written successfully." << endl;
        }
        CloseHandle(hFile);
    }

```

```

void openFileInNewWindow(const std::string& filename) {
    std::string command = "start " + filename;
    if (system(command.c_str()) == -1) {
        cout << "Failed to open file." << endl;
    }
}

```

```

void deleteFile(const std::string& filePath) {
    if (DeleteFileA(filePath.c_str()) == 0) {
        cerr << "Failed to delete file. Error code: " << GetLastError() <<
endl;
    }
    else {
        cout << "File deleted successfully." << endl;
    }
}

```

```
}
```

```
void deleteDirectory(const std::string& dirPath) {  
    char* cstr = new char[dirPath.length() + 1];  
    std::strcpy(cstr, dirPath.c_str());  
    if (!RemoveDirectoryA(cstr)) {  
        cerr << "Failed to delete directory. Error code: " <<  
GetLastError() << endl;  
    }  
    else {  
        cout << "Directory deleted successfully." << endl;  
    }  
    delete[] cstr;  
}
```

```
void renameFileOrFolder(const std::string& oldName, const  
std::string& newName) {  
    try {  
        std::filesystem::rename(oldName, newName);  
        cout << "Renamed successfully." << endl;  
    }  
    catch (const std::filesystem::filesystem_error& e) {  
        cout << "Error renaming: " << e.what() << endl;  
    }  
}
```

```
void waitForKeypress() {
```

```

    cout << "Press any key to continue...";
    _getch();
}

```

```

int main() {
    MenuOption selected = MenuOption::Option1;
    bool quit = false;

    while (!quit) {
        system("cls"); // Clear the console screen

        SetColor(14); // Yellow text
        cout << "\n";
        SetColor(11); // Cyan text
        cout << "*** Select an option:                **\n";
        cout << "*** 1. Create Directory          " << (selected ==
MenuOption::Option1 ? "<--" : "") << " **\n";
        cout << "*** 2. Create File              " << (selected ==
MenuOption::Option2 ? "<--" : "") << " **\n";
        cout << "*** 3. Delete Directory          " << (selected ==
MenuOption::Option3 ? "<--" : "") << " **\n";
        cout << "*** 4. Delete File              " << (selected ==
MenuOption::Option4 ? "<--" : "") << " **\n";
        cout << "*** 5. List Directory Contents  " << (selected ==
MenuOption::Option5 ? "<--" : "") << " **\n";
        cout << "*** 6. Read File              " << (selected ==
MenuOption::Option6 ? "<--" : "") << " **\n";
    }
}

```

```

        cout << "*** 7. Rename File/Directory    " << (selected ==
MenuOption::Option7 ? "<--" : "") << " **\n";
        cout << "*** 8. Exit                    " << (selected ==
MenuOption::Option8 ? "<--" : "") << " **\n";
        cout << "*** 9. Help                    " << (selected ==
MenuOption::Option9 ? "<--" : "") << " **\n";
        SetColor(14); // Yellow text
        cout << "\n";

        int key = _getch();
        if (key == 224) { // Arrow keys
            key = _getch();
            if (key == 72) selected =
static_cast<MenuOption>((static_cast<int>(selected) - 1 + 10) % 10); // Up
            if (key == 80) selected =
static_cast<MenuOption>((static_cast<int>(selected) + 1) % 10); // Down
        }
        else if (key == 13) { // Enter
            switch (selected) {
                case MenuOption::Option1: {
                    wstring dirPath;
                    cout << "Enter directory path: ";
                    wcin >> dirPath;
                    createDirectory(dirPath);
                    waitForKeypress();
                    break;
                }

```

```
case MenuOption::Option2: {
    string filePath;
    cout << "Enter file path: ";
    cin >> filePath;
    createFile(filePath);
    waitForKeypress();
    break;
}

case MenuOption::Option3: {
    string dirPath;
    cout << "Enter directory path: ";
    cin >> dirPath;
    deleteDirectory(dirPath);
    waitForKeypress();
    break;
}

case MenuOption::Option4: {
    string filePath;
    cout << "Enter file path: ";
    cin >> filePath;
    deleteFile(filePath);
    waitForKeypress();
    break;
}

case MenuOption::Option5: {
    wstring dirPath;
    cout << "Enter directory path: ";
```

```
        wcin >> dirPath;
        openDirectory(dirPath);
        waitForKeypress();
        break;
    }
    case MenuOption::Option6: {
        string filePath;
        cout << "Enter file path: ";
        cin >> filePath;
        openFileInNewWindow(filePath);
        waitForKeypress();
        break;
    }
    case MenuOption::Option7: {
        string oldName, newName;
        cout << "Enter current name: ";
        cin >> oldName;
        cout << "Enter new name: ";
        cin >> newName;
        renameFileOrFolder(oldName, newName);
        waitForKeypress();
        break;
    }
    case MenuOption::Option8: {
        quit = true;
        break;
    }
}
```



```

        case MenuOption::Option9: {
            cout << "Help - File Explorer Instructions\n"
                << "1. Create Directory: Enter the path to create a new
directory.\n"
                << "2. Create File: Enter the path to create a new file.\n"
                << "3. Delete Directory: Enter the path to delete an
existing directory.\n"
                << "4. Delete File: Enter the path to delete an existing
file.\n"
                << "5. List Directory Contents: Enter the path to list all
contents of a directory.\n"
                << "6. Read File: Enter the path to open a file in a new
window.\n"
                << "7. Rename File/Directory: Enter the current and new
name to rename.\n"
                << "8. Exit: Exit the program.\n"
                << "9. Help: Show this help message.\n";
            waitForKeypress();
            break;
        }
        default:
            break;
    }
}

return 0;
}

```