# LABORATORY EXERCISE 1

# Directory Management System

**Learning Objectives**

* Perform basic file and directory operations in C++.
* Develop a menu-driven console application.
* Apply modular programming by organizing code into functions.
* Handle user input and implement basic error checking.
* Retrieve and manipulate directory attributes.
* Execute system commands within a C++ program.
* Debug and test a C++ application effectively.

**Prerequisite student experiences and knowledge**

Students should have:

* A solid foundation in C++ programming, including syntax, data types, and control structures (loops, conditionals).
* Experience with functions (defining, calling, and returning values).
* Familiarity with input/output operations (cin, cout).
* Basic knowledge of file handling in C++.
* Understanding of system commands (e.g., system("cls") for clearing the console).
* Debugging skills to identify and resolve errors in C++ code.

**Background**

This exercise introduces students to practical file system operations in C++, reinforcing fundamental programming concepts. The Directory Management System simulates real-world applications that interact with the file system, providing hands-on experience in system-level programming.

**Materials/Resources**

* IDE: DevC++, TurboC++, or any C++ compiler.
* Documentation: C++ file handling libraries (`<direct.h>`, `<filesystem>` in C++17).
* Word processing
* Internet access for referencing documentation and resources.
* GitHub accounts for version control of development.

**Problem**

Develop a console-based Directory Management System in C++ that allows users to:

1. List files in the current directory.
2. Create a new directory.
3. Change the working directory.
4. Exit the program.

**Laboratory Activity**

**Functional Requirements**

* 1. Main Menu
     + Display the following options:

[1] List Files

[2] Create Directory

[3] Change Directory

[4] Exit List Files

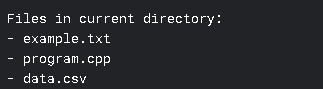
* 1. List File (Submenu)
* When the user selects List Files, Display:

[1] List All Files

[2] List Files by Extension (e.g., .txt)

[3] List Files by Pattern (e.g., "moha\*.\*")

Expected Output:



* 1. Create Directory
  + Prompt the user for a directory name.
  + If the directory does not exist, create it.
  + If it exists, display an error message.

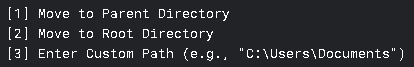
Expected Output:



Error Case:



* 1. Change Directory
  + Allow the User to:



Expected Output:



Error Case:



* 1. Exit

Terminate the program.

**Implementation Guidelines:**

Key Functions

1. `listFiles()` – Lists files based on user choice (all, by extension, or pattern).
2. `createDirectory()` – Creates a new directory with error handling.
3. `changeDirectory()` – Changes the working directory.
4. `mainMenu()` – Displays the main menu and handles user input.

Libraries to Use

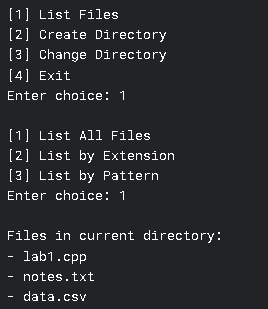
* `<iostream>` (Input/Output)
* `<direct.h>` (Directory operations in Windows)
* `<filesystem>` (C++17, cross-platform directory handling)
* `<string>` (String manipulation)

Error Handling

* Check if a directory exists before creating it.
* Validate user input for directory paths.
* Handle cases where directory changes fail.

**Sample Results**

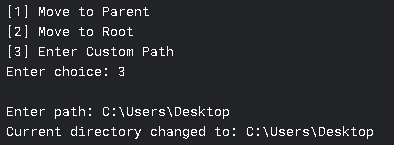
Sample Run 1: Listing Files



Sample Run 2: Creating a New Directory



Sample Run 3: Changing Directory



Sample Run 4: Changing the Working Directory

**Submission Requirements**

* C++ Source Code (`main.cpp`) – Well-commented and modular.
* GitHub Repository – Share the link in the submission.
* Screenshots – Demonstrate program functionality.

**QUESTIONS**

1. How does breaking the program into functions improve maintainability?
2. What issues arise when handling directories in C++? How can they be resolved?
3. Evaluate the usability of a console-based menu system.

**Flowchart**

**Output / Results**

Github link here: (HERE)

**Conclusion**