Coding Age

_____Practice Set: File Handling_____

Question Sets:

1. Create and Write to a File

Task: Write a program to create a file and write a user-provided string into it.

Explanation:

- The program should open a file in write (**w**) mode.
- If the file doesn't exist, it will be created automatically. If it exists, its content will be erased.
- The user provides a string that is written to the file using the **fprintf()** or **fputs()** function.
- After writing, the file is closed using **fclose().**

Input:

Enter the file name: example.txt
Enter the text to write: Hello, File Handling in C!

Output:

File written successfully.

---

2. Read Data from a File

Task: Read and display the content of a file.

Explanation :

- The program opens the file in read (**r**) mode. If the file does not exist, the program should handle the error gracefully.

- The content of the file is read using the **fgets()** function or **fscanf()** function, line by line or word by word.
- The program displays the content to the console.
- After reading, the file is closed to release resources.

Input:

Enter the file name to read: example.txt

Output:

Contents of the file:
Hello, File Handling in C!

---

3. Append Data to a File

Task: Add new data to an existing file without overwriting the previous content.

Explanation :

- Open the file in append (**a**) mode. This mode ensures new data is added to the end of the file without erasing existing content.
- Prompt the user to provide the data they wish to append.
- Use **fprintf()** or **fputs()** to append the data.
- After appending, close the file. If the file does not exist, it is created.

Input:

Enter the file name to append: example.txt
Enter the text to append: This is appended text.

Output:

Data appended successfully.

---

4. Count Characters, Words, and Lines in a File

Task: Count and display the number of characters, words, and lines in a file.

Explanation:

- Open the file in read mode.

- Read the file character by character using **fgetc()**.
- Count:
    - **Characters**: Increment the count for every character read (excluding EOF).
    - **Words**: Increment the count when a space or newline is encountered.
    - **Lines**: Increment the count when a newline (**\\n**) is encountered.
- Display the counts.

Input:

Enter the file name: example.txt

Output:

Characters: 50

Words: 8

Lines: 2

---

5. Copy Data from One File to Another

Task: Copy the contents of one file into another.

Explanation:

- Open the source file in read (**r**) mode and the destination file in write (**w**) mode.
- Read data from the source file using **fgetc()** or **fgets()** and write it to the destination file using **fputc()** or **fputs().**
- Ensure the file handles are closed after copying.
- Handle errors if the source file does not exist.

Input:

Enter the source file: example.txt

Enter the destination file: copy.txt

Output:

Data copied successfully from example.txt to copy.txt.

---

6. Delete a File

Task: Delete a specified file from the system.

Explanation:

- Use the **remove()** function to delete the file.
- Prompt the user for the file name.
- If the file is deleted successfully, print a success message. If the file doesn't exist, display an error message.

Input:

Enter the file name to delete: example.txt

Output:

File example.txt deleted successfully.

---

7. Count Occurrences of a Word in a File

Task: Find how many times a specific word appears in a file.

Explanation:

- Open the file in read mode.
- Read the file line by line or word by word using **fscanf()** or **fgets()**.
- Compare each word with the given word using **strcmp()** to count matches.
- Display the total count.

Input:

Enter the file name: textfile.txt
Enter the word to count: File

Output:

The word 'File' appears 3 times in the file.

---

8. Reverse File Content

Task: Reverse the content of a file and write it to a new file.

Explanation:

- Open the source file in read mode and read its content into a buffer (e.g., an array).
- Use string manipulation techniques to reverse the buffer content.
- Open a destination file in write mode and write the reversed content to it.
- Close both files.

Input:

Enter the source file: input.txt

Enter the destination file: reversed.txt

Output:

Content reversed and written to reversed.txt.

---

9. Merge Two Files

Task: Merge the contents of two files into a third file.

Explanation:

- Open both source files in read mode and the destination file in write mode.
- Read content from the first file and write it to the destination file.
- Repeat the same process for the second file.
- Ensure the contents are appended sequentially.
- Close all files after merging.

Input:

Enter the first file: file1.txt

Enter the second file: file2.txt

Enter the output file: merged.txt

Output:

Files merged successfully into merged.txt.

Projects

1. Project: Student Management System

Project Overview

The **Student Management System** is a console-based application designed to manage student records efficiently. It allows the user to perform operations like adding, viewing, searching, updating, and deleting student information. The project uses **file handling** to store the data permanently and ensures that the data is retrieved even after the program is closed.

System Features

1. **Add a New Student**:
    - Add a new student record, including ID, name, age, course, and enrollment status.
2. **View All Students**:
    - Display all stored student records in a tabular format.
3. **Search a Student by ID**:
    - Search for a student using their unique ID and display their details.
4. **Update Student Details**:
    - Modify details such as name, age, and course of an existing student.
5. **Delete a Student Record**:
    - Remove a student's record permanently using their ID.
6. **Mark Enrollment Status**:
    - Update a student's enrollment status (e.g., "Enrolled" or "Withdrawn").
7. **Exit Program**:
    - Close the application safely.

Menu Options

The program will present the following menu to the user:

1. Add Student
2. View All Students
3. Search Student by ID
4. Update Student Details
5. Delete Student
6. Mark Enrollment Status
7. Exit

---

2. Project: Advanced Employee Management System

Project Overview

The **Employee Management System** is a console-based application designed to handle various operations related to employee records. The project uses **file handling** in C to store employee data permanently. It provides features like adding employees, searching for employees by multiple criteria, updating records, generating reports, and managing salaries.

This project focuses on applying concepts like **modular programming**, **data structures**, and **file handling** in a real-world application.

System Features

1. **Add Employee**:
    - Add a new employee with ID, name, department, designation, salary, and contact details.
2. **View All Employees**:
    - Display all employee records in a tabular format.
3. **Search Employees**:
    - Search employees by ID, name, or department.
4. **Update Employee Details**:
    - Modify employee information like designation, salary, or contact details.
5. **Delete Employee**:
    - Remove an employee record permanently using their ID.
6. **Salary Management**:
    - View and update employee salary details.
7. **Generate Reports**:
    - Generate a summary report of employees by department or salary range.
8. **Sort Employees**:
    - Sort employee records by name, ID, or salary.
9. **Exit Program**:
    - Safely close the application.

Menu Options

The program will present the following menu to the user:

1. Add Employee
2. View All Employees
3. Search Employee by ID/Name/Department
4. Update Employee Details
5. Delete Employee
6. Manage Employee Salaries
7. Generate Reports (e.g., Department-wise or Salary-wise)
8. Sort Employees (e.g., by Name, ID, Salary)
9. Exit
