Assignment 10

Topics: File Access Methods and Command Line Arguments

Section A10.1: [File Access Methods]

A10.1a: Write a program to open a text file in **read mode** and display its content line by line.

Instructions:

1. Use `fopen()` to open the file in `"r"` mode.
2. Use `fgets()` in a loop to read each line from the file and print it.
3. Ensure that the file is properly closed using `fclose()`.

Example:

Input (file content):
Hello, this is line 1.
This is line 2.

Output:
Hello, this is line 1.
This is line 2.

Hint: Handle cases where the file doesn't exist by checking if the file pointer is `NULL`.

A10.1b: Write a program to **write data** to a text file using **write mode (`"w"`)**. If the file exists, its contents will be overwritten.

Instructions:

1. Use `fopen()` in `"w"` mode to create or overwrite the file.
2. Accept multiple lines of input from the user and write them to the file using `fprintf()`.
3. Close the file using `fclose()`.

Example:

Input: Write two lines to the file:
Line 1: Hello World
Line 2: C Programming

Output (in file):
Hello World
C Programming

Hint: The `"w"` mode overwrites the file, so any existing content is lost.

A10.1c: Write a program to **append data** to an existing file using **append mode (`"a"`)**.

Instructions:

1. Open the file in `"a"` mode.
2. Append additional lines of text provided by the user.
3. Ensure that the previous content is preserved, and new content is appended at the end.

Example:

Input (file already contains):
This is existing content.

Additional input (to append):
This is appended content.

Output (in file):
This is existing content.
This is appended content.

Hint: The `"a"` mode appends data at the end of the file, without overwriting existing content.

***A10.1d*: Write a program to **randomly access** a specific part of a file using `fseek()` and `ftell()`.**

Instructions:

1. Open a file in `"r"` mode.
2. Use `fseek()` to move to a specific byte position in the file.
3. Use `ftell()` to display the current file position.
4. Read and display the content starting from that position.

Example:

Input (file content): Hello World
Seek position: 6
Output: World

Hint: Use `fseek(file, offset, SEEK_SET)` to move the file pointer and `ftell()` to display the current pointer position.

A10.1e: Write a program to **copy binary data** from one file to another using **binary file access mode**.

Instructions:

1. Open a binary file in `"rb"` mode and another file in `"wb"` mode for writing.
2. Use `fread()` and `fwrite()` to copy the content.
3. Close both files after copying.

Example:

Input (binary file): [Binary Data]
Output (copied to another file): [Same Binary Data]

Hint: Binary file access allows copying raw data, unlike text files which interpret content as characters.

Section A10.2: [Command Line Arguments]

A10.2a: Write a program that takes a file name as a **command line argument** and displays the file's content.

Instructions:

1. Access the file name from `argv[]`.
2. Open the file in `"r"` mode.
3. Use `fgets()` to read the file and display its contents.
4. Handle errors in case the file is not found or cannot be opened.

Example:

Command: ./program file.txt
Output (content of file.txt):
This is the content of file.txt.

Hint: Use `argc` to ensure that the correct number of arguments is provided.

A10.2b: Write a program that accepts two numbers from the **command line arguments** and performs basic arithmetic operations (addition, subtraction, multiplication, division).

Instructions:

1. Extract the two numbers from `argv[]` and convert them to integers using `atoi()`.
2. Perform the four basic arithmetic operations (add, subtract, multiply, divide).
3. Display the results.

Example:

Command: ./program 10 5
Output:
Addition: 15
Subtraction: 5

Multiplication: 50
Division: 2

Hint: Use `argc` to verify that the correct number of arguments is provided.

A10.2c: Write a program that **counts the number of words** in a file provided via a **command line argument**.

Instructions:

1. Use `argv[]` to accept the file name.
2. Open the file in `"r"` mode.
3. Read through the file, count the number of words, and display the count.
4. Close the file.

Example:

Command: ./program textfile.txt
Output: Number of words = 50

Hint: Use space, newline, and tab characters as word delimiters to count the words.

A10.2d: Write a program that accepts two file names via **command line arguments**. The program should **copy the contents** of the first file to the second file.

Instructions:

1. Use `argv[]` to get the source and destination file names.
2. Open the source file in `"r"` mode and the destination file in `"w"` mode.
3. Copy the contents from the source to the destination file.
4. Close both files after copying.

Example:

Command: ./program source.txt destination.txt
Output: Contents of source.txt copied to destination.txt

Hint: Handle error checking to ensure both files are opened successfully.

Section A10.3: [Combining File Operations, CLA, DMA, and Structures]

A10.3a: Write a program that accepts a file name from the **command line**, reads the file, and **dynamically allocates memory** to store its contents. After storing the contents, print the file's data.

Instructions:

1. Use `argv[]` to get the file name from the command line.
2. Open the file in `"r"` mode.
3. Dynamically allocate memory using `malloc()` to store the file's contents.
4. Read the file's content into the allocated memory and print it.
5. Free the allocated memory after use.

Example:

Command: ./program file.txt
Output (file.txt content): "Hello, this is file content."

Hint: Use `fseek()` and `ftell()` to determine the file size for allocating memory.

A10.3b: Write a program that accepts a file name and an integer `N` from the **command line**. The program should read the first `N` lines from the file and store them in a **dynamically allocated array of strings**. Display the lines on the screen.

Instructions:

1. Use `argv[]` to get the file name and the integer `N` from the command line.
2. Open the file in `"r"` mode.
3. Dynamically allocate memory for storing `N` lines of text.
4. Read the first `N` lines into the dynamically allocated array and display them.
5. Free the allocated memory after use.

Example:

Command: ./program file.txt 3
Output (first 3 lines of file.txt):
Line 1: ...
Line 2: ...
Line 3: ...

Hint: Use `malloc()` to allocate memory for each line and `fgets()` to read each line from the file.

A10.3c: Write a program to manage a list of students, each with a `name`, `roll_number`, and `marks`. Accept the file name via **command line arguments**, load student records from the file into a **dynamically allocated array of structures**, and display the records.

Instructions:

1. Define a `Student` structure with fields: `name`, `roll_number`, and `marks`.
2. Use `argv[]` to get the file name from the command line.
3. Open the file in `"r"` mode.

4. Dynamically allocate memory for an array of `Student` structures based on the number of records in the file.
5. Load the student data from the file into the structure array and display the records.
6. Free the allocated memory after use.

Example:

Command: ./program students.txt
Output:
Student 1: Name: John, Roll Number: 101, Marks: 85
Student 2: Name: Alice, Roll Number: 102, Marks: 92

Hint: Use `fscanf()` to read the data from the file into the structure array.

A10.3d: Write a program that accepts two file names from the **command line**. The program should load student records from the first file, sort the records by `marks` in descending order, and save the sorted records to the second file.

Instructions:

1. Define a `Student` structure with fields: `name`, `roll_number`, and `marks`.
2. Use `argv[]` to get the source and destination file names from the command line.
3. Load student records from the source file into a dynamically allocated array of structures.
4. Sort the array based on the `marks` field.
5. Save the sorted records to the destination file.
6. Free the allocated memory after use.

Example:

Command: ./program students.txt sorted_students.txt
Output (in sorted_students.txt):
Alice, Roll Number: 102, Marks: 92
John, Roll Number: 101, Marks: 85

Hint: Use a sorting algorithm (e.g., bubble sort) to sort the array and `fprintf()` to write the sorted data to the destination file.