Lecture 08 - File Handling in C Programming

Q1

What is the usage of files for data storage in C programming?

Files are commonly used for data storage in C programming for a wide range of applications and use cases. Here are some of the key usages of files for data storage in C programming:

- Data Persistence: Files allow data to be stored persistently. Data written to a file can be retained even after a program terminates, making it available for future use.
- Database Systems: Databases often use files for data storage. While database management systems (DBMS) provide high-level abstractions for data storage, they ultimately write data to files on disk.
- Archiving and Backups: Files are used to archive data, enabling the retention of historical records. Regular backups of data are stored in files to ensure data integrity and recovery in case of system failures.
- Inter-process Communication: Files can be used for communication between different processes or programs. Data can be written to a file by one process and read by another, enabling data sharing.
- Custom File Formats: C programs often define their custom file formats for specific data storage needs. This allows them to structure and organize data as required.

In summary, the usage of files for data storage in C programming is versatile and essential. Files are a fundamental component of data management, data persistence, and communication in various applications, from simple configuration files to complex database systems and data-intensive batch processing tasks.

Q2

What kind of modes can be used to open a new file in C language?

In C, you can open a file for different purposes, such as reading, writing, or appending, by specifying the appropriate file mode when using the fopen function. The file mode is specified as the second argument to fopen. Here are some common file modes and their purposes:

Read ("r"):

Purpose: Opens a file for reading.

Behavior: If the file doesn't exist, it returns NULL. You can read from the file, but you cannot write to it.

Example: fopen("myfile.txt", "r");

Write ("w"):

Purpose: Opens a file for writing.

Behavior: If the file already exists, it truncates its content. If the file doesn't exist, it creates a new empty file.

Example: fopen("myfile.txt", "w");

Append ("a"):

Purpose: Opens a file for appending.

Behavior: If the file exists, it appends data to the end. If the file doesn't exist, it creates a new file.

Example: fopen("myfile.txt", "a");

Read and Write ("r+"):

Purpose: Opens a file for both reading and writing.

Behavior: The file must exist. You can read and write data in the file.

Example: fopen("myfile.txt", "r+");

Write and Read ("w+"):

Purpose: Opens a file for both writing and reading.

Behavior: If the file exists, it truncates its content. You can read and write data in the file.

Example: fopen("myfile.txt", "w+");

Append and Read ("a+"):

Purpose: Opens a file for both appending and reading.

Behavior: If the file exists, it appends data to the end. You can read and write data in the file.

Example: fopen("myfile.txt", "a+");

Q3

How to create a new file in C to store data?

To create a new file in C to store data, you can use the fopen function with the appropriate mode. Here's a step-by-step guide on how to create a new file in C and write data to it:

Declare a FILE* pointer:

Declare a pointer of type FILE* to hold the reference to the file.

```
FILE *file_ptr;
```

Open the file for writing:

Use the fopen function to create a new file and open it for writing. Provide the filename and specify the mode as "w" (write). If the file already exists, this will truncate it. If it doesn't exist, a new empty file will be created.

```
file_ptr = fopen("newfile.txt", "w");
if (file_ptr == NULL) {
    printf ("Unable to open the file");
    return 1; // Exit the program with an error code.
}
```

Write data to the file:

Use functions like fprintf to write data to the file. Here's an example using fprintf to write text to the file:

```
fprintf(file_ptr, "This is the first line of text.\n");
fprintf(file_ptr, "This is the second line of text.\n");
```

Close the file:

After you've finished writing data to the file, it's important to close it using the fclose function. This step releases system resources and ensures that any buffered data is written to the file.

```
fclose(file_ptr);
```

Error Handling:

Always check for errors when working with files, as the fopen function can fail.

```
if (file_ptr == NULL) {
    printf("Error opening file");
    return 1;
}
```

Here's a complete example of creating a new file, writing data to it, and then closing it:

```
#include <stdio.h>
int main() {
    FILE *file_ptr;

    // Open the file for writing
    file_ptr = fopen("newfile.txt", "w");
    if (file_ptr == NULL) {
        printf("Unable to open the file");
        return 1;
    }

    // Write data to the file
    fprintf(file_ptr, "This is the first line of text.\n");
    fprintf(file_ptr, "This is the second line of text.\n");

    // Close the file
    fclose(file_ptr);
```

```
    return 0;
}
```

This code creates a new file named "newfile.txt," writes two lines of text to it, and then closes the file.

Q4

How to read file content in C ?

You can read the content of a file in C by using functions like fopen, fscanf. Here's a step-by-step guide on how to read file content in C:

Declare a FILE* pointer:

Declare a pointer of type FILE* to hold the reference to the file.

```
FILE *file_ptr;
```

Open the file for reading:

Use the fopen function to open an existing file for reading. Provide the filename and specify the mode as "r" (read).

```
file_ptr = fopen("myfile.txt", "r");
if (file_ptr == NULL) {
    printf ("Unable to open the file");
    return 1; // Exit the program with an error code
}
```

Read data from the file:

Use functions like fscanf to read data from the file. Here's an example using fscanf to read lines of text from the file:

```
scanf(file_ptr, "%s", buffer) //Reading the first line
while (!file_ptr(feof)) {
```

```
        // Process the line of text in 'buffer'
        printf("%s", buffer); // Print the line to the console
        scanf(file_ptr, “%s”, buffer)
    }
```

Close the file:

After you've finished reading data from the file, it's important to close it using the fclose function to release system resources.

```
    fclose(file_ptr);
```

Error Handling:

Always check for errors when working with files.

```
    if (file_ptr == NULL) {
        printf ("Error opening file");
        return 1;
    }
```