Unit 9:

Data Files in C Programming

Data files in C are used to store and retrieve data permanently. Unlike variables and arrays, which store data temporarily in RAM, files store data on secondary storage devices like hard drives, allowing data persistence even after program termination.

---

1. Introduction to Data Files

In C programming, a data file is a collection of related information stored on disk. It enables a program to read and write data for long-term use.

Why Use Files?

- Data remains saved even after the program terminates.
- Large amounts of data can be stored and processed efficiently.
- Helps in handling structured data (e.g., records, reports, logs).

2. Types of Files (Text File & Binary File)

1. Text Files

Definition

A text file is a file that stores data in a human-readable format using ASCII or Unicode characters. Each line in a text file is terminated with a newline (`\n`) character. Text files are commonly used for storing structured data such as CSV, logs, and configuration files**.**

Characteristics

- Stores data as plain text (readable by humans).
- Uses characters like alphabets, numbers, and symbols.
- Each line ends with a newline character (`\n`).
- Can be opened and edited with any text editor (e.g., Notepad, VS Code).
- Requires formatting when storing numerical or structured data.
- Slower access speed compared to binary files due to character encoding and decoding.

2. Binary Files

Definition

A binary file stores data in machine-readable format (0s and 1s). Instead of storing human-readable characters, it saves data in raw memory format, which is faster and more space-efficient.

Characteristics

- Stores data in binary format (not human-readable).
- More efficient in terms of speed and memory.
- Can store any data type (integers, floats, structs, etc.) without conversion.
- Cannot be opened or edited directly using a text editor.
- Must be accessed programmatically using functions like `fread()` and `fwrite()`.

Feature	Text File	Binary File
Definition	Stores data as human-readable text (ASCII/Unicode).	Stores data in machine-readable binary format (0s and 1s).
Storage	Uses characters for storage (e.g., numbers as "123").	Stores raw memory data (e.g., numbers as binary 01111011).
Readability	Can be opened in Notepad or any text editor.	Cannot be opened in a text editor (displays garbage values).
File Size	Larger (extra storage for character encoding, spaces, and line breaks).	Smaller (stores data directly in memory format).
Speed	Slower due to conversion between characters and data types.	Faster because no conversion is needed.
Data Integrity	Data may be lost or misinterpreted (e.g., newline characters can change formatting).	No data loss since raw bytes are stored directly.
Usage	Suitable for logs, reports, and CSV files.	Suitable for databases, images, and executables.

3. File Handling Operations in C

File handling in C allows us to create, open, read, write, and close files. It is done using the **FILE** pointer and functions from the `<stdio.h>` library.

---

4. Opening and Closing a File in C

Opening a File

To open a file, we use the `fopen()` function. It returns a pointer of type `FILE*`.

Syntax of `fopen()`

```
FILE *ptr;
ptr = fopen("filename", "mode");
```

- `"filename"`: The name of the file (with extension).
- `"mode"`: Specifies the purpose (read, write, append, etc.).

File Opening Modes in C

Mode	Description

`"r"`	Opens an existing file for reading. Returns NULL if the file does not exist.
`"w"`	Opens a file for writing. If the file exists, it is overwritten. If it doesn't exist, a new file is created.
`"a"`	Opens a file for appending (writing at the end). If the file doesn't exist, it creates a new one.
`"r+"`	Opens a file for both reading and writing. File must exist.
`"w+"`	Opens a file for both reading and writing. If the file exists, it is overwritten. If it doesn't exist, a new file is created.
`"a+"`	Opens a file for reading and appending. Creates a file if it does not exist.

Example: Opening a File for Writing

```
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "w"); // Open file in write mode
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    fprintf(file, "Hello, File Handling in C!\n"); // Write to file
    fclose(file); // Close the file
    return 0;
}
```

Closing a File

After performing file operations, we must close the file using `fclose()`.
This prevents data loss and frees system resources.

Syntax of `fclose()`

```
fclose(file_pointer);
```

Example: Closing a File

```
FILE *file = fopen("example.txt", "r");
if (file != NULL) {
    fclose(file); // Closing the file
}
```

5 Creating a File in C

To create a new file, we use the `"w"` (write) or `"a"` (append) mode with `fopen()`.
If the file does not exist, it will be created.

Example: Creating a File in Write Mode

```
#include <stdio.h>

int main() {
    FILE *file = fopen("newfile.txt", "w"); // Creates a new file if it does not exist
    if (file == NULL) {
        printf("File creation failed!\n");
        return 1;
    }
    printf("File created successfully!\n");
    fclose(file); // Close file after creation
    return 0;
}
```

Example: Creating a File in Append Mode

```
#include <stdio.h>

int main() {
    FILE *file = fopen("appendfile.txt", "a"); // Opens existing file or creates new one
    if (file == NULL) {
        printf("File creation failed!\n");
        return 1;
    }
    printf("File ready for appending!\n");
    fclose(file);
    return 0;
}
```

Summary of File handling operation

1. **Opening a file**: Use `fopen("filename", "mode")`.
2. **Closing a file**: Use `fclose(file_pointer)`.
3. **Creating a file**: Use `"w"` or `"a"` mode in `fopen()`, which creates a file if it doesn't exist

6. Library Functions for File Handling in C

C provides various functions to read from and write to files. These functions help handle text and binary files efficiently. Below is a detailed explanation of each function.

Writing to a File in C

1. `fputs()` – Write a String to a File

- Used to write a string to a file.

- Does not append a newline (`\n`) automatically.

Syntax:

```
int fputs(const char *str, FILE *stream);
```

Example:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "w");
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    fputs("Hello, this is a text file.\n", file);
    fputs("Learning file handling in C.", file);
    fclose(file);
    return 0;
}
```

Output in `example.txt`:

```
Hello, this is a text file.
Learning file handling in C.
```

2. `fputc()` – Write a Single Character to a File

- Used to write a single character to a file.

Syntax:

```
int fputc(int ch, FILE *stream);
```

Example:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("example.txt", "w");
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    fputc('H', file);
    fputc('i', file);
    fputc('!', file);
    fclose(file);
    return 0;
}
```

Output in `example.txt`:

Hi!

3. `fprintf()` – Write Formatted Data to a File

- Works like `printf()` but writes formatted data to a file.

Syntax:

```
int fprintf(FILE *stream, const char *format, ...);
```

Example:

```
#include <stdio.h>
int main() {
    FILE *file = fopen("data.txt", "w");
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    int age = 21;
    float marks = 95.5;
    fprintf(file, "Name: Alice\nAge: %d\nMarks: %.2f", age, marks);
    fclose(file);
    return 0;
}
```

Output in `data.txt`:

```
Name: Alice

Age: 21

Marks: 95.50
```

Reading from a File in C

4. `fgets()` – Read a String from a File

- Reads a line (string) from a file.

- Stops at a newline (`\n`) or EOF.

Syntax:

```
char *fgets(char *str, int n, FILE *stream);
```

Example:

```
#include <stdio.h>
int main() {
    char buffer[50];
    FILE *file = fopen("data.txt", "r");
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    while (fgets(buffer, 50, file)) { // Reads line by line
        printf("%s", buffer);
    }
    fclose(file);
    return 0;
}
```

✅ Output:

```
Name: Alice
Age: 21
Marks: 95.50
```

---

5. `fgetc()` – Read a Single Character from a File

- Reads one character at a time.

Syntax:

```
int fgetc(FILE *stream);
```

Example:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("data.txt", "r");
```

```
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    char ch;
    while ((ch = fgetc(file)) != EOF) { // Read character by character
        printf("%c", ch);
    }
    fclose(file);
    return 0;
}
```

✅ Output (same as file content):

```
Name: Alice

Age: 21

Marks: 95.50
```

---

6. `fscanf()` – Read Formatted Data from a File

- Works like `scanf()` but reads from a file.

Syntax:

```
int fscanf(FILE *stream, const char *format, ...);
```

Example:

```
#include <stdio.h>

int main() {
    FILE *file = fopen("data.txt", "r");
    if (file == NULL) {
        printf("Error opening file!\n");
        return 1;
    }
    char name[20];
    int age;
    float marks;

    fscanf(file, "Name: %s\nAge: %d\nMarks: %f", name, &age, &marks);
    printf("Name: %s\nAge: %d\nMarks: %.2f\n", name, age, marks);
```

```
    fclose(file);
    return 0;
}
```

✅ **Output:**

```
Name: Alice

Age: 21

Marks: 95.50
```

Summary Table of File Handling Functions in C

Function	Purpose
Writing Functions	
fputs()	Writes a string to a file.
fputc()	Writes a single character to a file.
fprintf()	Writes formatted data to a file (like printf).
Reading Functions	
fgets()	Reads a string (line) from a file.
fgetc()	Reads a single character from a file.
fscanf()	Reads formatted data from a file (like scanf).