Module 2 – Introduction to Programming

1) Overview of C Programming:

THEORY EXERCISE:

- Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.

Ans.

C is a procedural programming language initially developed by Dennis Ritchie in 1972.

It was mainly developed as a system programming language to write the UNIX operating system.

C program has a primary function that must be named main. The main function serves as the starting point for program execution.

- The main features of the C language include:
  - General Purpose and Portable
  - Low-level Memory Access
  - Fast Speed
  - Clean Syntax
  - ❖ C is a procedural language that supports structured programming. Its static system and compiler are written in C.
  - ❖ C programs are highly portable because C compilers are available for almost all hardware and operating systems.
  - ❖ C is widely used in embedded systems, such as microcontrollers, robotics, and IoT (Internet of Things) devices.

❖ LAB EXERCISE:

- Research and provide three real-world applications where C programming is extensively used, such as embedded systems, operating systems, or game development.

Ans.

1. Embedded Systems:

Application Example: Automotive Control Systems.

Embedded systems are specialized computing systems that do not look like traditional computers but are designed to perform specific tasks.

Real-World Example: In automotive control systems, C is used to program the embedded microcontrollers that manage engine functions, safety features, and infotainment systems.
These systems require real-time, efficient, and low-latency performance, which C provides.

2. Operating Systems:

Application Example: Linux Kernel

Operating systems form the backbone of computing devices, providing the necessary infrastructure for applications to run, manage hardware, and facilitate communication between software and hardware.

Real-World Example: The Linux kernel, one of the most popular and widely used open-source operating systems, is primarily written in C.
It provides a foundation for various applications and is used in environments ranging from smartphones to supercomputers.

3. Game Development:

Application Example: Game Engines.

Game development requires high performance, especially for graphics rendering, real-time processing, and interaction with hardware.
Game engines are the frameworks used to develop games. They provide tools for rendering graphics, handling physics, and managing game logic.

Real-World Example: Unreal Engine, one of the most widely used game engines for AAA and indie games, is developed using C and C++.
Games like Fortnite and PUBG were built using Unreal Engine, relying on C for key parts of their performance-heavy engine.

2) Setting Up Environment:

THEORY EXERCISE:

- Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like Dev C++, VS Code, or Code Blocks.

Ans.

GCC (GNU Compiler Collection) is a popular C compiler.

GCC is a widely used C compiler. To install GCC, the steps vary depending on your operating system.

Step 1: Install GCC (GNU Compiler Collection):

1) Download MinGW (Minimalist GNU for Windows):

- Go to MinGW-w64 and download the installer.
- Choose the correct version (e.g., mingw-w64-install.exe for 32/64-bit).

2)Install MinGW:

- Run the installer and follow the setup wizard. Select the appropriate architecture and posix for the thread model.
- Make sure to select mingw32-gcc-g++ (for the C and C++ compiler) during installation.

3)Set Up Environment Variables:

- After installation, go to Control Panel > System and Security > System > Advanced system settings.
- Click on Environment Variables and under System Variables, find Path.
- Edit the Path and add the path to the bin folder of MinGW.

4)Verify the Installation:

- Open a Command Prompt and type gcc --version to check if GCC is installed properly.

Step 2: Install an IDE (Dev C++, VS Code, or Code::Blocks):

1: Dev C++ (Windows):

1. Download Dev C++:
   - Go to Dev C++ official website and download the latest version of Dev C++.
2. Install Dev C++:
   - Run the installer and follow the on-screen instructions.
3. Configure Compiler (If necessary):
   - In Dev C++, go to Tools > Compiler Options and ensure that the MinGW GCC compiler is selected.

2: Visual Studio Code (VS Code):

1. Download VS Code:
   - Go to VS Code's official website and download the appropriate version for your operating system.
2. Install VS Code:
   - Run the installer and follow the on-screen instructions.
3. Install C/C++ Extension:

- Open VS Code and go to the Extensions view (Ctrl + Shift + X).
- Search for "C/C++" and install the extension provided by Microsoft.

3: Code::Blocks:

1. Download Code::Blocks:
   - Go to Code::Blocks download page and download the version with MinGW bundled (for Windows).
2. Install Code::Blocks:
   - Run the installer and select the appropriate options.
3. Verify GCC Compiler:
   - Open Code::Blocks and go to Settings > Compiler.
   - Ensure that the GCC compiler is selected in the Selected Compiler dropdown.

3. Basic Structure of a C Program:

THEORY EXERCISE:

➢ Explain the basic structure of a C program, including headers, main functions, comments, data types, and variables. Provide examples.

Ans.

1. Headers:

Headers are files that provide declarations of functions and macros that can be used in your program.

These are often libraries provided by C, like stdio.h (for standard input/output), or user-defined header files.

Example:

```
#include <stdio.h>
```

2. Main Function:

The main function is the entry point of any C program.

When the program is executed, the execution starts from the main() function.

Example:

```
int main() {
        return 0;
       }
```

3. Comments:

Comments are used to explain the code and make it more readable.
They are ignored by the compiler.

Example:

1)Single-line comments:

```
// This is a single-line comment.
```

2)Multi-line comments:

```
/* This is a multi-line comment
that spans multiple lines */
```

4. Data Types:

C provides several built-in data types to declare variables.
The data type defines the type of data a variable can hold.

Common data types in C:

int: For integers (whole numbers).
Float: For floating-point numbers (decimal numbers).
Char: For individual characters.
Double: For double-precision floating-point numbers.
Void: For functions that do not return a value.

Example:

```
int age = 25; // integer
float height = 5.9; // floating point
char grade = 'A'; // character
```

5. Variables:

Variables are containers for storing data.
They are declared by specifying the data type and the variable name.

Example:

```
Int age = 25; // Declare an integer variable called age and assign a value of 25.
Float height = 5.9; // Declare a float variable called height and assign a value of 5.9.
Char grade = 'A'; // Declare a char variable called grade and assign a value of 'A'.
```

4. Operators in C:

THEORY EXERCISE:

- Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

Ans.

1. Arithmetic Operators:

These operators perform mathematical calculations like addition, subtraction, multiplication, division, and modulus.

- +: Addition
  Example: a + b
- -: Subtraction
  Example: a - b
- *: Multiplication
  Example: a * b
- /: Division
  Example: a / b
- %: Modulus (remainder of division)
  Example: a % b

2. Relational Operators:

These operators are used to compare two values.

- ==: Equal to
  Example: a == b (true if a is equal to b)
- !=: Not equal to
  Example: a != b (true if a is not equal to b)
- >: Greater than
  Example: a > b (true if a is greater than b)
- <: Less than
  Example: a < b (true if a is less than b)
- >=: Greater than or equal to
  Example: a >= b (true if a is greater than or equal to b)
- <=: Less than or equal to
  Example: a <= b (true if a is less than or equal to b)

3. Logical Operators:

Logical operators are used to perform logical operations, often in conditions or loops.

They combine multiple conditions and return true or false.

- &&: Logical AND
  Example: a > 0 && b < 5 (true if both conditions are true)
- ||: Logical OR
  Example: a > 0 || b < 5 (true if at least one condition is true)
- !: Logical NOT
  Example: !(a > b) (true if the condition a > b is false)

4. Assignment Operators:

Assignment operators are used to assign values to variables.

- =: Simple assignment
  Example: a = 5; (assigns the value 5 to a)
- +=: Add and assign
  Example: a += 5; (adds 5 to a and assigns the result to a)
- -=: Subtract and assign
  Example: a -= 5; (subtracts 5 from a and assigns the result to a)
- *=: Multiply and assign
  Example: a *= 5; (multiplies a by 5 and assigns the result to a)
- /=: Divide and assign
  Example: a /= 5; (divides a by 5 and assigns the result to a)
- %=: Modulus and assign
  Example: a %= 5; (replaces the remainder of a divided by 5)

5. Increment/Decrement Operators:

These operators are used to increase or decrease the value of a variable by one.

- ++: Increment operator
  Example: a++ (increments the value of a by 1)
  Example: ++a (increments the value of a by 1 before using it in the expression)
- --: Decrement operator
  Example: a-- (decrements the value of a by 1)
  Example: --a (decrements the value of a by 1 before using it in the expression)

6. Bitwise Operators:

Bitwise operators are used to perform operations on the individual bits of numbers.

- &: Bitwise AND
  Example: a & b (performs a bitwise AND on a and b)
- |: Bitwise OR
  Example: a | b (performs a bitwise OR on a and b)
- ^: Bitwise XOR (exclusive OR)
  Example: a ^ b (performs a bitwise XOR on a and b)
- ~: Bitwise NOT (one's complement)
  Example: ~a (flips the bits of a)
- <<: Left shift
  Example: a << 2 (shifts the bits of a left by 2 positions)
- >>: Right shift
  Example: a >> 2 (shifts the bits of a right by 2 positions)

7. Conditional (Ternary) Operator:

The conditional operator is a shorthand for an if-else statement. It evaluates a condition and returns one value if true, and another if false.

- ?: : Conditional (Ternary) Operator
  Example: result = (a > b) ? a : b;
  If a > b is true, the result will be assigned a.
  Otherwise, the result will be assigned b.

5. Control Flow Statements in C:

THEORY EXERCISE:

➢ Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

Ans.

1. if Statement:

The if statement executes a block of code only if the specified condition is true.

Example:

```
#include <stdio.h>
int main()
{
    int a = 10;
```

```
        if (a > 5)
    {
        printf("\n\n\t a is greater than 5 ");
    }
    }
```

2. else Statement:

The else statement follows an if statement and is executed when the condition in the if is false.

Example:

```
#include <stdio.h>
int main()
{
    int x;
    printf("\n\n\t enter the num=");
    scanf("%d",&x);

    if (x > 5)
    {
    printf("\n\n\t x is greater than 5\n");
    } else
    {
    printf("\n\n\t x is less than or equal to 5\n");
    }
}
```

3. Nested if-else:

This is a chain of if-else statements used to check multiple conditions.
It is also called a "ladder" because each condition is checked in sequence.

Example:

```
#include <stdio.h>
int main()
{
int x;
    printf("\n\n\t enter the num=");
    scanf("%d",&x);
if (x > 30) {
```

```
 printf("\n\n\t x is greater than 30\n");
} else if (x > 10) {
printf("\n\n\t x is greater than 10 but less than or equal to 30\n");
} else {
 printf("\n\n\t x is less than or equal to 10\n");
}
}
```

4. switch Statement:

The switch statement allows you to test a variable against several possible values, each corresponding to a case.

If a match is found, the corresponding block of code is executed.

Example:

```
#include <stdio.h>
int main() {
Int day;
printf("\n\n\t enter the 1to5 num=");
scanf("%d",&day);

switch (day) {
        case 1:
                printf("Monday\n");
                break;
        case 2:
                printf("Tuesday\n");
                break;
        case 3:
                printf("Wednesday\n");
                break;
        case 4:
                printf("Thursday\n");
                break;
        case 5:
                printf("Friday\n");
                break;
        default:
                printf("Invalid day\n");
```

```
}
}
```

6. Looping in C:

THEORY EXERCISE:

- Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.

Ans.

1. while loop:

Syntax:

```
while (condition) {
        // Statements to be executed
            }
```

The condition is evaluated before the loop body executes.
If the condition is false initially, the loop's body is never executed.

Example:

```
#include<stdio.h>
main()
{
    int i = 1;
    while (i <= 5)
    {
        printf("%d ", i);
        i++;
    }
}
```

2. for loop:

Syntax:

```
for (initialization; condition; increment/decrement)
    {
        // Statements to be executed
    }
```

The for loop combines initialization, condition checking, and increment/decrement in a single line.
The loop executes as long as the condition is true.

Example:

```
#include<stdio.h>
main()
	{
	int i=1;
	for (int i = 1; i <= 5; i++)
	{
		printf("%d ", i);
	}
```

3. do-while loop:

Syntax:

```
	do
	{
		// Statements to be executed
	} while (condition);
```

The body of the do-while loop is executed at least once, and then the condition is checked.

If the condition is true, the loop repeats; if false, the loop terminates.

Example:

```
#include<stdio.h>
int i = 1;
do
{
	printf("%d ", i);
	i++;
} while (i <= 5);
```

most appropriate:

While:

When the number of iterations is unknown.

When you need to continue as long as a condition is true.

If the loop might not execute at all if the condition is false.

For:

When you know the number of iterations ahead of time.

Ideal for iterating through ranges, arrays, or lists.

Do-while:

When the loop body must run at least once, even if the condition is false initially.

Useful for user input validation or menu-driven programs.

7. Loop Control Statements

THEORY EXERCISE:

- Explain the use of break, continue, and go to statements in C. Provide examples of each.

Ans.

1. break Statement:

The break statement is used to immediately exit from a loop(for, while, do-while) or a switch statement.

Use cases:

To exit a loop early based on a condition.

To exit a switch statement once a match is found.

Example:

```
#include <stdio.h>
int main()
{
Int i=1;
for (i = 1; i <= 10; i++)
{
if (i == 5) {
                    break;  // Exits the loop when i is 5
                }
                    printf("%d ", i);
}
                    printf("\nLoop exited.");
                    return 0;
}
```

2. continue Statement:

The continue statement is used to skip the current iteration of a loop and move to the next iteration.

Use cases:

To skip certain iterations in a loop based on a condition.

Example:

```
#include <stdio.h>
int main()
{
Int i=1;
```

```
for (i = 1; i <= 10; i++)
{
if (i == 5) {
                continue; // Skips the iteration when i is 5
            }
                printf("%d ", i);
}
                printf("\nLoop exited.");
                return 0;
}
```

3. goto Statement:

The goto statement is used to transfer control to another part of the program, specified by a label.

Use cases:

Rarely used in modern structured programming, but can be useful for handling exceptional cases or breaking out deeply nested loops.

Example:

```
#include <stdio.h>
int main()
{
    int i = 1;
start:
if (i > 10)
{
        goto end;  // Jumps to the 'end' label
}
        printf("%d ", i);
        i++;
        goto start;  // Jumps to the 'start' label
end:
        printf("\nLoop ended.");
        return 0;
}
```

8. Functions in C

THEORY EXERCISE:

- What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

Ans.

In C programming, a function is a block of code that performs a specific task.
Functions help in organizing the code into smaller, reusable pieces.
They allow you to break a large program into smaller, manageable parts, making the code easier to read, debug, and maintain.

Parts of a Function:

A function in C has the following components:

1. Function Declaration (or Function Prototype)
2. Function Definition
3. Function Call

1. Function Declaration:

A function declaration tells the compiler about the function's name, its return type, and the types of its parameters.
The declaration does not contain the function's body; it simply provides information for the compiler to understand how the function should be used.

Syntax:

```
return_type function_name(parameter_list);
```

- return_type: The type of value the function returns (e.g., int, float, void if it doesn't return anything).
- function_name: The name of the function.
- parameter_list: A list of input parameters that the function takes (can be empty if no parameters).

Example:

```
Int add(int, int);  // Declaration of a function named "add" that returns an integer and takes two integers as parameters.
```

2. Function Definition:

A function definition provides the actual implementation of the function.
It consists of the function's body, where the task is performed.

Syntax:

```
return_type function_name(parameter_list)
{
        // function body
}
```

- return_type: Same as in the declaration (e.g., int, void).
- function_name: The name of the function.
- parameter_list: Parameters the function accepts, matching the declaration.

Example:

```
int add(int a, int b)
{
    Return a + b;  // Function body adds two numbers and returns the result.
}
```

In the above definition:

- add is the function's name.
- int a, int b are the parameters (input).
- a + b is the operation performed in the function body.

3. Function Call:

A function call is how you execute the function in the program.
You call the function by specifying its name, followed by parentheses containing any arguments it requires.

Syntax:

```
function_name(arguments);
```

- function_name: The name of the function to call.
- arguments: Values passed to the function, corresponding to the function's parameters.

Example:

```
Int result = add(5, 3);  // Calling the function "add" with arguments 5 and 3
```

Example for declaration, definition, call:

```
int maximum();
main()
{
```

```
        int max=maximum();
        printf("\n\n\t maximum num=%d",max);

}
int maximum()
{
        int x, y;

        printf("\n\n\t enter the first num=");
        scanf("%d",&x);
        printf("\n\n\t enter the second num=");
        scanf("%d",&y);

        if(x>y)
        return x;
        else
                return y;
}
```

9. Arrays in C:

THEORY EXERCISE:

- Explain the concept of arrays in C. Differentiate between one-dimensional and multi-dimensional arrays with examples.

Ans.

Concept of Arrays in C:

In C, an array is a collection of elements of the same type stored in contiguous memory locations.

The elements of an array are indexed, starting from 0 for the first element, 1 for the second, and so on.

Key points:

Array Declaration:

The array is declared by specifying the type of its elements and the number of elements it can hold.

Syntax:

```
type arrayName[arraySize];
```

- type: The data type of the elements (e.g., int, float, char).
- arrayName: The name of the array.
- arraySize: The number of elements in the array.

Example of Array Declaration:

```
int arr[5];  // Declares an array of 5 integers.
```

One-Dimensional Array:

A one-dimensional array is the simplest form of an array where the data is stored in a single row or line.
It's essentially a list of values.

Syntax:

```
dataType arrayName[arraySize];
```

Example:

```
#include<stdio.h>
main()
{
    // Declare and initialize a one-dimensional array
    int arr[5]={10, 20, 30, 40, 50};

    printf("\n\n\t arr[0] : %d", arr[0]);
    printf("\n\n\t arr[1] : %d", arr[1]);
    printf("\n\n\t arr[2] : %d", arr[2]);
    printf("\n\n\t arr[3] : %d", arr[3]);
    printf("\n\n\t arr[4] : %d", arr[4]);
}
```

Multi-Dimensional Arrays:

A multi-dimensional array is an array that has more than one dimension.
The most common form is the two-dimensional array, which is like a matrix (rows and columns).

Syntax:

```
dataType arrayName[rows][columns];
```

rows is the number of rows, and columns is the number of columns in the 2D array.

Example of 2D array dimensional:

```
#include<stdio.h>
main()
{
        int mat[3][3]={10, 20, 30, 40, 50, 60, 70, 80, 90};
        int r, c;

for(r=0;r<3;r++)
{
        for(c=0;c<3;c++)
        {
                printf(" %d", mat[r][c]);
        }
        printf("\n");
}
}
```

10. Pointers in C:

THEORY EXERCISE:

- Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

Ans.

A pointer in C is a variable that stores the memory address of another variable.
Pointers must be declared before they can be used, just like a normal variable.

There are 2 important operators that we will use in pointers concepts i.e.

- Dereferencing operator(*) used to declare pointer variable and access the value stored in the address.
- Address operator(&) used to return the address of a variable or to access the address of a variable to a pointer.

1. Declaring Pointers:

To declare a pointer, you specify the type of data it points to, followed by an asterisk (*), which indicates it's a pointer:
Example:

```
int *ptr; // Pointer to an integer
```

```
char *ptr; // Pointer to a character
```

2. Initializing Pointers:

A pointer is initialized with the address of a variable. To get the address of a variable, you use the address-of operator (&).

Example:

```
int x = 10;
int *ptr = &x; // ptr now holds the address of variable x
```

Why Pointers are Important in C:

Memory management:

Pointers enable efficient memory management and dynamic memory allocation and deallocation.

Data structures:

Pointers are essential for creating complex data structures like linked lists, trees, and graphs.

Code reduction:

Pointers can reduce code and improve performance.

Passing data:

Pointers can be used to pass data efficiently between functions.

Returning values:

Pointers can be used to return values from functions.

11. Strings in C:

THEORY EXERCISE:

- Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

Ans.

1. strlen() – Calculate the Length of a String:

The strlen() function is used to calculate the length of a string, excluding the null terminator ('\0').

It returns the number of characters in the string.

Example:

```
#include <stdio.h>
#include <string.h>
```

```
int main()
{
    char str[] = "Hello, world!";
    printf("Length of the string is:\n", strlen(str));
}
```

2. strcpy() – Copy a String:

The strcpy() function is used to copy the contents of one string into another. It includes the null terminator when copying.

Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char src[] = "Hello, world!";
    char dest[20];

    strcpy(dest, src);
    printf("Destination string: %s\n", dest);
}
```

3. strcat()--concatenates two strings:

It appends the contents of one string (source) to the end of another (destination). The destination string must have enough space to accommodate the new content.

Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[50] = "Hello, ";
    char str2[50] = "World!";
    strcat(str1, str2);
    printf("\n\n\t Concatenated string: %s\n", str1);
    // Output: Hello, World!
    return 0;
}
```

4. strcmp()--compare two strings:

It compares two strings lexicographically (i.e., character by character).
The function returns an integer based on the comparison.
Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
        char str1[] = "Apple";
        char str2[] = "Banana";
        int result = strcmp(str1, str2);
        if (result < 0)
                printf("\n\n\t str1 is less than str2\n");
        else if (result > 0)
                printf("\n\n\t str1 is greater than str2\n");
        else
                printf("\n\n\t str1 and str2 are equal\n");
        return 0;
}
```

5. strchr()--Search for Character:

It searches for the first occurrence of a character in a string.
It returns a pointer to the first occurrence of the character c in the string str.
If the character is not found, it returns NULL.
Example:

```
#include <stdio.h>
#include <string.h>
int main()
{
        // Declare a string
        char str[] = "Hello, World!";

        // Search for the first occurrence of the character 'W'
        char *ptr = strchr(str, 'W');

        if (ptr != NULL)
```

```
    {
    printf("Character 'W' found at position: %ld\n", ptr - str);
    }
    else
    {
    printf("Character 'W' not found in the string.\n");
     }
    return 0;
}
```

12. Structures in C:

THEORY EXERCISE:

- Explain the concept of structures in C. Describe how to declare, initialize, and access structure members.

Ans.

Structure is a user-defined data type that allows grouping different data types under a single name.
The members of a structure can be of different data types, and each member can be accessed individually.
Structures are used to represent real-world entities with multiple attributes.

Syntax of a Structure Declaration:

```
struct structure_name
{
data_type member1;
data_type member2;
// Add more members as needed
};
```

Declaring a Structure:

Declare a structure to represent a Student with three members: name (a string), age (an integer), and grade (a character):

Syntax:

```
struct Student
{
char name[50];    // Array of characters (string)
int age;         // Integer
char grade;      // Character
```

};

Initialize a Structure:

During Declaration (at the time of declaring the structure variable): You can provide initial values for the members.

Syntax:

Struct Student student1 = {"John Doe", 20, 'A'};

Accessing Structure Members:

You access structure members using the dot notation (.).

For example:

- student1.name will access the named member of the student1 structure.
- student1.age will access the age member of student1.

13. File Handling in C:

THEORY EXERCISE:

- Explain the importance of file handling in C. Discuss how to perform file operations like opening, closing, reading, and writing files.

Ans.

File handling in C is the process in which we create, open, read, write, and close operations on a file.

C language provides different functions such as fopen(), fwrite(), fread(), fseek(), fprintf(), etc.

To perform input, output, and many different C file operations in our program.

Types of Files in C:

- Text Files.
- Binary Files.

Text Files:

A text file contains data in the form of ASCII characters and generally stores a stream of characters.

- It can be read or written by any text editor.
- They are generally stored with a **.**txt file extension.

- Text files can also be used to store the source code.

Binary Files:

A binary file contains data in binary form (i.e. 0's and 1's) instead of ASCII characters.

They contain data that is stored similarly to how it is stored in the main memory.

- More secure as they are not easily readable.
- They are generally stored with a **.**bin file extension.

Basic File Operations in C:

There are four basic file operations you can perform in C:

1. Opening a file (fopen())
2. Closing a file (fclose())
3. Reading from a file (fscanf(), fgets(), fread())
4. Writing to a file (fprintf(), fputs(), fwrite())

1. Opening a File: fopen():

To open the file in mode (w, r, a),

Syntax:

```
FILE *fopen(const char *filename, const char *mode);
```

Filename: Name of the file to open.

Mode: Specifies the file access mode.

Common modes include:

- "r": Read-only mode (file must exist).
- "w": Write-only mode (creates a new file, or overwrites if file exists).
- "a": Append mode (writes data at the end of the file).

2. Closing a File: fclose():

To close the file from the current mode.

Syntax:

```
int fclose(FILE *file);
```

Example:

```
fclose(file);  // Close the file when done
```

3. Reading from a File:fgets():

To read the data from the file.

Syntax:

```
char *fgets(char *str, int num, FILE *file);
```

4. Writing to a File:fputs():

To write the contents into the file.

Syntax:

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