Module 2 – Introduction to Programming

1. Overview of C Programming

❖ THEORY EXERCISE:

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

Ans: **History and Development of C**

- The origins of C can be traced back to the early **1970s at Bell Labs**. **Dennis Ritchie**, alongside **Brian Kernighan** and others, developed C as an evolution of previous languages like B and BCPL. The primary motivation behind C's creation was to build a language that could be used to develop the Unix operating system, which was initially written in assembly language.
- In 1972, C emerged as a language with features that made it suitable for low-level programming, while also supporting structured programming, a growing trend at the time.
- In 1978, the release of the book *"The C Programming Language"* by Brian Kernighan and Dennis Ritchie, often referred to as "K&R C," played a critical role in popularizing the language. The book served as both a tutorial and a reference manual, helping spread C beyond Bell Labs.
- The **ANSI C standard** (American National Standards Institute), developed in 1989 and known as **C89**, standardized the language, ensuring code portability across different systems. Later updates, including **C99**, **C11**, and **C18**, introduced new features like inline functions, variable-length arrays, multithreading support, and improved Unicode handling, ensuring that C kept evolving with changing programming needs.

❖ The History and Evolution of C Programming

- The C programming language holds a foundational place in the world of computer science and software development. Created over five decades ago, C has not only withstood the test of time but has also laid the groundwork for many modern programming languages.

- Its simplicity, efficiency, and power make it an enduring choice for system-level programming and application development. This essay explores the history and evolution of C, its importance, and the reasons why it continues to be widely used today.

Importance of C Programming

1. **Foundation of Operating Systems:**
    - Most operating systems, including Unix, Linux, and parts of Windows, were written in C. The language provided the right mix of low-level memory access and high-level constructs.
2. **Language Influence:**
    - C has heavily influenced many other programming languages, including C++, Java, C#, Objective-C, and even newer languages like Rust and Go. Its syntax and structure serve as a blueprint for these languages.
3. **System Programming:**
    - Due to its direct memory manipulation and performance efficiency, C remains the language of choice for developing embedded systems, device drivers, and system software.
4. **Portability and Efficiency:**
    - C programs are highly portable, which means they can run on different machines with minimal changes. Its compiled nature also makes it faster than most interpreted languages.

Why C Is Still Used Today

1. **Performance:**
    - C offers high performance and minimal runtime overhead, making it ideal for applications where speed is critical.
2. **Control Over System Resources:**
    - C allows direct manipulation of memory through pointers, which is essential in systems programming and embedded environments.
3. **Legacy Systems:**

- A vast amount of legacy code is written in C. Maintaining and upgrading this software requires knowledge of the language.

4. **Learning Foundation:**
    - C is often taught in academic settings because it helps students understand fundamental programming concepts, including data structures, memory management, and algorithms.
5. **Embedded Systems**
    - C dominates the embedded systems industry due to its small footprint and efficiency. It is used in microcontrollers, firmware, robotics, and other hardware-level applications.

2. Setting Up Environment

❖ THEORY EXERCISE:

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

Ans: **Steps to Install GCC and Set Up an IDE**

➢ **Install GCC Compiler**

- **Windows:** Download *MinGW* from mingw.org, install, and add bin folder path to Environment Variables.
- **Mac:** Install Xcode Command Line Tools using xcode-select --install.
- **Linux:** Install via terminal: sudo apt install build-essential (Ubuntu/Debian) or equivalent for your distro.

➢ **Install IDE**

- **Dev-C**++: Download from SourceForge, install, open, and set compiler path in *Tools → Compiler Options*.
- **VS Code:** Download from code.visualstudio.com, install, then add the *C/C++ Extension* from Extensions Marketplace.
- **Code:** Blocks: Download from codeblocks.org (with MinGW), install, and start coding.

➢ **Verify Setup**

- Open terminal/command prompt, run:
  - ✓ **gcc –version**

3. Basic Structure of a C Program

❖ THEORY EXERCISE:

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

Ans: Basic structure of a C program step-by-step so it's easy to understand.

➢ Header Files

- Header files tell the compiler what libraries or predefined functions your program will use. They're included at the top with the **#include** directive.
- **Example: #include <stdio.h>** // Standard input-output functions (printf, scanf),

  #include<conio.h> // Console **I**nput/**O**utput header file functions(clrscr, getch).

➢ Main Function

- Every C program must have a main() function. Execution starts here.
- The entry point of the program, written as int main(), where execution begins.
- **Example:** int main()

```
{
printf("Hello, World!\n");
return 0;
 }
```

➢ Comments

- In C, comments are used to add explanatory notes within the code, which are ignored by the compiler. There are two types of comments.

1. **Single-line comments:**
    - ✓ Use **//** to comment out a single line
    - ✓ **Example: // Hello World!**
2. **Multi-line comments:**
    - ✓ Enclosed between **/* and */,** used for comments spanning multiple lines.
    - ✓ **Example: /* Welcome to C Program */**

- Comments help improve code readability and provide explanations for complex sections.

➢ Data Types

- A data type specifies what type of data a variable can store such as integer, floating, character etc.
- There are many tyes of data types of C.
    - ▪ **Basic Data Types**
        - o **Integer(int)**
            - ✓ The integer datatype in C is used to store the integer numbers (any number including positive, negative and zero without decimal part).
            - ✓ **Size: 2 Bytes**
            - ✓ **Format Specifier: %d**
            - ✓ **Example: int x=10;**
        - o **Float(float)**
            - ✓ float data type is used to store single precision floating-point values. These values are decimal and exponential numbers.
            - ✓ **Size: 4 Bytes**
            - ✓ **Format Specifier: %f**
            - ✓ **Example: float y=20;**
        - o **Double(double)**
            - ✓ The double data type in C is used to store decimal numbers (numbers with floating point values) with double precision. It can

easily accommodate about 16 to 17 digits after or before a decimal point.

- ✓ **Size: 8 Bytes**
- ✓ **Format Spefier: %lf**
- ✓ **Example: double data=1.565545;**

- ○ **Char(char)**
  - ✓ Used to store single characters. Can be signed or unsigned.
  - ✓ **Size: 1 Byte**
  - ✓ **Forarmat Spefier: %c**
  - ✓ **Example: char d = ”A”;**
- ○ **Boolean(bool)**
  - ✓ It is reture a value true and false;
  - ✓ **Size: 1 Byte**
  - ✓ **Example: bool f=true;**
- ○ **Void(void)**
  - ✓ Void is an empty data type that has no value.
  - ✓ **It has no return value.**
  - ✓ **Example: void main() { }**

- ▪ **Derived Data Types**
  - ○ **Array**
    - ✓ Array is a group of data that share the common name.
    - ✓ Array index is always start from 0.
    - ✓ Array is provides squencial data.
    - ✓ There are two types of Arrray.
      1. **One diamenstional Array**
      2. **Two diamenstional Array**
  - ○ **Pointer**
    - ✓ A pointer is declared by specifying the data type of the variable it will point to, followed by an asterisk (*) and the pointer's name. It is then initialized with

the address of a variable using the address-of operator (&).

- **Example:**
  ```
  int num = 10;
  int *ptr = &num;
  ```

- ### Structure
  - In C programming, a structure (struct) is a user-defined data type that allows for the grouping of variables of different data types under a single name. This enables the creation of complex data types that represent real-world entities with multiple attributes.
  - **Syntax:**
    ```
    struct StructureName {
       data_type member1;
       data_type member2;
       ...
       data_type memberN;
        };
    ```
  - **Example:**
    ```
    struct student {
          char name[50];
          int roll_number;
          float gpa;
       };
    ```

- ### Union
  - A union is declared similarly to a structure. Provide the name of the union and define its member variables:
  - **Syntax:**
    ```
    union union_name{
       type1 member2;
       type2 member3;
       type3 member3;
       .    .

    } variable_name;
    ```
  - **Example:**
    ```
    union Student {
    ```

```
        int rollNo;
        float height;
        char firstLetter;
    };   union Student data;
```

- **Variables**
  - Variable is a data name any it is used to store the data value.
  - Variables are named memory locations to store values. They **must** be declared with a type before use.
  - **Example: int x=50;**

- **Example:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
   // Find  a maximum number
   int no1,no2,no3;
   clrscr();
   printf("\n Enter the Number1:");
   scanf("%d",&no1);
   printf("\n Enter the Number2:");
   scanf("%d",&no2);
   printf("\nEnter the Number3:");
   scanf("%d",&no3);
   if(no1>no2 && no1>no3)
   {
      printf("\n Maximum Number1 => %d:",no1);
   }
   else if(no2>no1 && no2>no3)
   {
```

```
  printf("\n Maximum Number2 => %d",no2);
}
else
{
  printf("\n Maximum Number3 => %d:",no3);
}
getch();

}
```

4. Operators in C

❖ THEORY EXERCISE:

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

Ans: Operators are the basic components of C programming. They are symbols that represent some kind of operation, such as mathematical, relational, bitwise, conditional, or logical computations, which are to be performed on values or variables.

❖ Types of Operators

➢ Arithmatic Operator

- The arithmetic operators are used to perform arithmetic/mathematical operations on operands.

Operator	Function	Example
+	**Addition**	**var=a+b**
-	**Subtraction**	**var=a-b**
*****	**Multification**	**var=a*b**
/	**Division**	**var=a/b**
%	**Modulo**	**var=a%b**

➢ Relational Operator

- The relational operators in C are used for the comparison of the two operands. All these operators are binary operators that return true or false values as the result of comparison.

Operator	Meaning of Operator	Example
==	**Equal to**	**5 == 3 is evaluated to 0**
>	**Greater than**	**5 > 3 is evaluated to 1**
<	**Less than**	**5 < 3 is evaluated to 0**
!=	**Not equal to**	**5 != 3 is evaluated to 1**
>=	**Greater than or equal to**	**5 >= 3 is evaluated to 1**
<=	**Less than or equal to**	**5 <= 3 is evaluated to 0**

➢ Logical Operator

- Logical Operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either true or false.

Operator	Meaning	Example
&&	**Logical AND. True only if all operands are true**	**If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0**
||	**Logical OR. True only if either one operand is true**	**If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1**

!	**Logical NOT. True only if the operand is 0**	**If c = 5 then, expression !(c==5) equals to 0.**

- **Bitwise Operator**
  - The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands.

Operators	**Meaning of operators**
&	**Bitwise AND**
|	**Bitwise OR**
^	**Bitwise exclusive OR**
~	**Bitwise complement**
<<	**Shift left**
>>	**Shift right**

- **Assignment Operator**
  - **The Basic type of Assignment operator is '='.**
  - **There are other derived operators.**

  Example: *=, -=, /=, +=

- **Increment/Decrement Operator**
  - **It is increament of one and decreament of 1.**

5. Control Flow Statements in C

❖ THEORY EXERCISE:

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

Ans: In C programming, decision-making statements allow the program to choose different actions based on conditions.

- ➢ **The main are types of statements:**
  - **if Statement**
    - ▪ **Executes a block of code only if the condition is true.**
    - ▪ **Syntax:**
      ```
      if (condition) {
         // Code if condition is true
      }
      ```
  - **if-else**
    - ▪ Executes one block if the condition is true, otherwise executes another block.
    - ▪ **Syntax:**
      ```
      if (condition) {
         // Code if true
      } else {
         // Code if false
      }
      ```
  - **Nested if-else**
    - ▪ When an if or else contains another if-else for multiple conditions.
    - ▪ **Syntax:**
      ```
      if (condition1) {
         if (condition2) {
            // Code if both true   } else {
            // Code if condition1 true but condition2 false
      }} else {
         // Code if condition1 false
      }
      ```
  - **Ladder if-else Statement**
    - ▪ if-else-if ladder, this is used when you have multiple conditions to check one after another.
    - ▪ **Syntax:**
      ```
      if (condition1) {
         // Code for condition1
      } else if (condition2) {
      ```

```
    // Code for condition2
} else if (condition3) {
    // Code for condition3
} else {
    // Code if none are true  }
```

6. Looping in C

❖ THEORY EXERCISE:

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

Ans: In C, loops are used to repeat a block of code until a condition is met.

The three main loops are:

- ➢ **while loop**
    - Repeats code while a condition is true. The condition is checked before the first iteration (entry-controlled**.**
    - **Syntax: while (condition) { // code }**
    - **Example: #include <stdio.h>**
      int main() { int i = 1; while (i <= 5) { printf("%d ", i);
      i++; } return 0; }
- ➢ **for loop**
    - A compact loop where initialization, condition, and update are all in one line. Also entry-controlled.
    - **Syntax: for (initialization; condition; update) {**
      // code}
    - **Example: #include <stdio.h>**

      int main() { for (int i = 1; i <= 5; i++) { printf("%d ", i);

      } return 0; }
- ➢ **do-while loop**
    - Similar to while, but the condition is checked after executing the loop body (exit-controlled loop).

- **Syntax: do {   // code } while (condition);**
- **Example: #include <stdio.h>**
  int main() {   int i = 1;   do {   printf("%d ", i);   i++;
  } while (i <= 5);   return 0; }

7. Loop Control Statements

❖ THEORY EXERCISE:

1: Explain the use of break, continue, and goto statements in C. Provide examples of each.

Ans: In C, jump statements are used to alter the normal flow of program execution.

1. **Break**
   - Immediately terminates the nearest enclosing loop (for, while, do-while) or switch statement.
   - Control moves to the statement after the loop or switch.
   - **Syntax: break;**
   - **Example: #include <stdio.h>**
   - **int main() {   int i;   for (i = 1; i <= 5; i++) {**
     if (i == 3) {
     break; // Exit loop when i is 3   }
     printf("%d ", i);   }   return 0; }
2. **Continue**
   - Skips the rest of the code in the current iteration of the loop and moves to the next iteration.
   - **Syntax: continue;**
   - **Example: #include <stdio.h>**
   - **int main() {   int i;**
     for (i = 1; i <= 5; i++) {   if (i == 3) {
     continue; // Skip printing 3   }   printf("%d ", i) }
     return 0; }
3. **goto**
   - Transfers control unconditionally to a labeled statement in the same function.

- Can move both forward and backward in code (but generally discouraged due to readability issues)
- **Syntax:** goto label_name; // ... label_name: // code
- **Example: #include <stdio.h>**
- **int main() { int i = 1; start: printf("%d ", i); i++; if (i <= 5) { goto start; // Jump to the label 'start' } return 0; }**

8. Functions in C

❖ THEORY EXERCISE:

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

Ans: A function in C is a block of code that performs a specific task, can be reused, and can be called from other parts of the program.

➢ **Types of Functions**

a. **Library Functions – Predefined in C standard library (e.g., printf(), scanf(), strlen()).**
b. **User-Defined Functions – Created by the user.**

➢ **Function Declaration (Prototype)**

- Tells the compiler the function name, return type, and parameters before its first use.
- Allows the function to be called before it is defined.
- **Syntax: return_type function_name(parameter_list);**
- **Example: int add(int a, int b);**

➢ **Function Definition**

- Contains the actual code (body) that executes when the function is called.
- **Syntax: return_type function_name(parameter_list) { // statements return value; // if return_type is not void }**
- **Example: int add(int a, int b) { return a + b;**

```
}
```

- **Calling a Function**
  - To use a function, write its name followed by parentheses containing any required arguments.
  - Control transfers to the function; after execution, control returns to the calling point.
  - **Syntax: function_name(arguments);**
  - **Example: #include <stdio.h>**

```
// Function Declaration

int add(int a, int b); int main() {   int num1 = 5, num2 = 10, result

   // Function Call
  result = add(num1, num2);
     printf("Sum = %d\n", result);    return 0; }
  // Function Definition
   int add(int a, int b) {
     return a + b; }
```

9. Arrays in C

❖ THEORY EXERCISE:

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

Ans: **Differentiate between one-dimensional and multi-dimensional arrays**

Parameters	**One-Dimensional**	**Multi-Dimensional**
Definition	Stores elements in a single row.	Stores elements in multiple rows/columns.
Syntax	**data_type array_name[size];**	**data_type array_name[rows][columns];**
Dimensions	A one-dimensional array has only one dimension.	A two-dimensional array has a total of two dimensions.

Representation	Represent multiple data items as a list.	Represent multiple data items as a table consisting of rows and columns.
Size(bytes)	size of(datatype of the variable of the array) * size of the array	size of(datatype of the variable of the array) the number of rows the number of columns.
index	Single index (e.g., arr[i]).	Multiple indexes (e.g., arr[i][j]).

Example	**#include <stdio.h>**
	int main() {
	int arr[5] = {10, 20, 30, 40, 50};
	for (int i = 0; i < 5; i++) {
	printf("%d ", arr[i]);
	}
	return 0;
	}
		#include <stdio.h>
		int main() {
		int matrix[2][3] = {
		{1, 2, 3},
		{4, 5, 6}
		};
		for (int i = 0; i < 2; i++) {
		for (int j = 0; j < 3; j++) {
		printf("%d ", matrix[i][j]);
		}
		printf("\n");
		}
		return 0;
		}

10. Pointers in C

❖ THEORY EXERCISE:

1: 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. They are declared with *, initialized with &, and used with * (dereference operator). They are crucial in C for efficiency, dynamic memory, and advanced data structures.

- **Normally, variables store values.**
- **A pointer stores the *address* of a value.**

- **Declaration of Pointers**
  - Pointers are declared using the * (asterisk) symbol.
- **Syntax: data_type *pointer_name;**
- **Example:**

```
#include <stdio.h>
 int main() {
         int x = 10;
         int *p;      // declare pointer
          p = &x;      // initialize with address of x
          printf("Value of x: %d\n", x);       // 10
   printf("Address of x: %p\n", &x);
   printf("Pointer p holds: %p\n", p);   // same address
   printf("Value at address stored in p: %d\n", *p); // 10

  return 0;

}
```

- **Why are Pointers Important in C?**
  - Pointers are a powerful feature in C for several reasons:

1. **Direct Memory Access**
   - Pointers allow accessing and modifying memory directly.
   - Useful for system-level programming (e.g., OS, device drivers).
2. **Efficient Function Arguments**
   - Passing large structures/arrays by pointer is faster than copying the whole data.
   - Example: Passing arrays to functions.

3. **Dynamic Memory Allocation**

- Functions like malloc(), calloc(), free() in <stdlib.h> use pointers to manage memory at runtime.

4. **Data Structures**

- Pointers are the foundation of linked lists, trees, graphs, stacks, queues, etc.

5. **Pointer Arithmetic**

- Useful for navigating arrays and memory blocks.

6. **Flexibility**

- Allows functions to modify actual variables (pass-by-reference behavior).

11. Strings in C

❖ THEORY EXERCISE:

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

Ans: **String Handling Function in C**

➢ **strlen() – String Length**

- Returns the length of a string (number of characters excluding the null '\0').
- **Syntax: size_t strlen(const char *str);**
- **Example: #include <stdio.h> #include <string.h>**

```
    int main() {
         char str[] = "Hello World";
           printf("Length of string = %lu\n", strlen(str));
            return 0;  }  // Output:  Length of string = 11
```

➢ **strcpy() – Copy String**

- Copies one string into another.
- **Syntax: char* strcpy(char *destination, const char *source);**
- **Example: #include <stdio.h> #include <string.h>**

```
int main() {
    char src[] = "C Programming";
    char dest[50];
    strcpy(dest, src);
       printf("Copied string: %s\n", dest);
      return 0;   } // Output: Copied string: C Programming
```

- **strcat() – Concatenate Strings**
  - Concatenates (joins) two strings. The second string is appended to the first.
  - **Syntax: char* strcat(char *destination, const char *source);**
  - **Example: #include <stdio.h> #include <string.h>**

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

- **strcmp() – Compare Strings**
  - Compares two strings.
  - Return values
    - 0 → if both strings are equal
    - < 0 → if first string is less than second
    - > 0 → if first string is greater than second
  - **Syntax: int strcmp(const char *str1, const char *str2);**
  - **Example: #include <stdio.h> #include <string.h>**

    ```
    int main() {
      char str1[] = "Apple";
      char str2[] = "Banana";
      int result = strcmp(str1, str2);
       if (result == 0)
         printf("Strings are equal\n");
      else if (result < 0)
         printf("str1 is smaller than str2\n");
      else
    ```

```
  printf("str1 is greater than str2\n");
return 0; } // Output: str1 is smaller than str2
```

➢ **strchr() – Find Character in String**

- Finds the first occurrence of a character in a string.
- **Syntax: char* strchr(const char *str, int character);**
- **Example: #include <stdio.h> #include <string.h>**

```
int main() {
   char str[] = "Hello World";
   char *ptr = strchr(str, 'o');
    if (ptr != NULL)
      printf("First occurrence of 'o': %s\n", ptr);
   else
      printf("Character not found\n");
   return 0; } // Output: First occurrence of 'o': o World
```

12. Structures in C

❖ THEORY EXERCISE:

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

Ans: In C programming, a structure (struct) is a user-defined data type that allows for the grouping of variables of different data types under a single name. This enables the creation of complex data types that represent real-world entities with multiple attributes.

➢ **Syntax: struct StructureName {**

```
data_type member1;
data_type member2;
...
data_type memberN;      };
```

➢ **Example: #include <stdio.h> #include <string.h>**

```
  struct Student {
          int rollNo;
          char name[50];
          float marks;
       };
int main() {
   struct Student s1 = {101, "Amit", 87.5};  // initialization
   struct Student s2;   // another student
   s2.rollNo = 102;
   strcpy(s2.name, "Pooja");
   s2.marks = 92.0;

   // Print details
   printf("Student 1: %d, %s, %.2f\n", s1.rollNo, s1.name, s1.marks);
   printf("Student 2: %d, %s, %.2f\n", s2.rollNo, s2.name, s2.marks);

   return 0;

}
```

13. File Handling in C

❖ THEORY EXERCISE:

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

Ans: **<u>Importance of File Handling in C</u>**

- ➢ In C programming, when a program runs, data is usually stored in **RAM**, which is temporary. Once the program ends, the data is lost. To store data permanently, we use **files**.
- ➢ Permanent storage – Data is saved on disk, not lost after program termination.
- ➢ Large data management – Files can hold more data than variables/arrays.
- ➢ Data sharing – Multiple programs can access the same file.
- ➢ Better organization – Data can be structured into text or binary files.

- Flexibility – Allows reading, writing, appending, and modifying data easily.
- Input/Output operations – Provides a way to read data from and write data to external files.

File Operations in C

- The <stdio.h> library provides functions to work with files. Files are accessed using a file pointer of type FILE *.

1. **fopen() => Opening a File**
   - **We use fopen() to open a file.**
   - **File opening modes:**
     - "r" → read (file must exist).
     - "w" → write (creates new file or overwrites existing).
     - "a" → append (writes at end of file).
     - "r+" → read + write.
     - "w+" → read + write (overwrites).
     - "a+" → read + append.
   - Example: FILE *fp;
     fp = fopen("data.txt", "r"); // open file in read mode
2. **fclose() => Closing a File**
   - **After finishing, close the file to free resources:**
   - **Example: fclose(fp);**
3. **fprintf() => Writing to a File**
   - **We can write text or data to a file.**
   - **Using fprintf() (formatted writing):**
     - **Example: FILE *fp = fopen("data.txt", "w");**
       fprintf(fp, "Hello, this is a file handling example.\n");
       fclose(fp);
   - **Using fputs() (string writing):**
     - **Example: FILE *fp = fopen("data.txt", "a");**
       fputs("Appending new line.\n", fp);
       fclose(fp);
   - **Using fputc() (single character writing):**

- Example:
  ```
  FILE *fp = fopen("data.txt", "w");
  fputc('A', fp);
  fclose(fp);
  ```

4. fscanf() => Reading from a File
    - We can read stored data from a file.
    - Using fscanf() (formatted reading):
        - Example:
          ```
          FILE *fp = fopen("data.txt", "r");
              char str[50];
              fscanf(fp, "%s", str);   // reads a word
              printf("Read: %s", str);
              fclose(fp);
          ```
    - Using fgets() (read string/line):
        - Example:
          ```
          FILE *fp = fopen("data.txt", "r");
          char buffer[100];
          fgets(buffer, 100, fp);   // reads one line
          printf("Line: %s", buffer);
          fclose(fp);
          ```
    - Using fgetc() (read character):
        - Example:
          ```
          FILE *fp = fopen("data.txt", "r");
          char buffer[100];
          fgets(buffer, 100, fp);   // reads one line
          printf("Line: %s", buffer);
          fclose(fp);
          ```

❖ Example: Writing and Reading from a File

```
#include <stdio.h>
    int main() {
      FILE *fp;
      char str[100];

      // Writing to a file
      fp = fopen("sample.txt", "w");
```

```
      fprintf(fp, "Hello, C programming with files!\n");
      fclose(fp);

      // Reading from a file
      fp = fopen("sample.txt", "r");
      fgets(str, 100, fp);
      printf("File content: %s", str);
      fclose(fp);

    return 0;
}
```